10.0 BUS OPERATION

10.1 Data Transfer Mechanism

All data transfers occur as a result of one or more bus cycles. Logical data operands of byte, word and doubleword lengths may be transferred without restrictions on physical address alignment. Data may be accessed at any byte boundary but two or three cycles may be required for unaligned data transfers. (See section 10.1.2, ''Dynamic Data Bus Sizing,'' and section 10.1.5, ''Operand Alignment.'')

The Military Intel486 processor address signals are split into two components. High-order address bits are provided by the address lines, A2–A31. The byte enables, BE0#–BE3#, form the low-order address and provide linear selects for the four bytes of the 32-bit address bus.

The byte enable outputs are asserted when their associated data bus bytes are involved with the present bus cycle, as listed in Table 10-1. Byte enable patterns that have a negated byte enable separating two or three asserted byte enables will never occur (see Table 10-5). All other byte enable patterns are possible.

Table 10-1. Byte Enables and Associated Data and Operand Bytes

Byte Enable Signal	Associated Data Bus Signals	
BE0#	D0–D7	(byte 0–least significant)
BE1#	D8–D15	(byte 1)
BE2#	D16–D23	(byte 2)
BE3#	D24–D31	(byte 3–most significant)

Address bits A0 and A1 of the physical operand's base address can be created when necessary. Use of the byte enables to create A0 and A1 is shown in Table 10-2. The byte enables can also be decoded to generate BLE# (byte low enable) and BHE# (byte high enable). These signals are needed to address 16-bit memory systems. (See section 10.1.3, ''Interfacing with 8-, 16-, and 32-Bit Memories.'')

10.1.1 MEMORY AND I/O SPACES

Bus cycles may access physical memory space or I/O space. Peripheral devices in the system may either be memory-mapped, or I/O-mapped, or both. Physical memory addresses range from 00000000H to FFFFFFFFH (4 gigabytes). I/O addresses range from 00000000H to 0000FFFFH (64 Kbytes) for programmed I/O. (See Figure 10-1.)

Table 10-2. Generating A0–A31 from BE0#–BE3# and A2–A31

Military Intel486™ Processor Address Signals								
A31	**. . .**	**A2**			**BE3#**	**BE2#**	**BE1#**	**BE0#**
	Physical Base Address							
A31	. . .	A2	A1	A0				
A31	. . .	A2	0	0	X	X	X	Low
A31	. . .	A2	0	1	X	X	Low	High
A31	. . .	A2	1	0	X	Low	High	High
A31	. . .	A2	1	1	Low	High	High	High

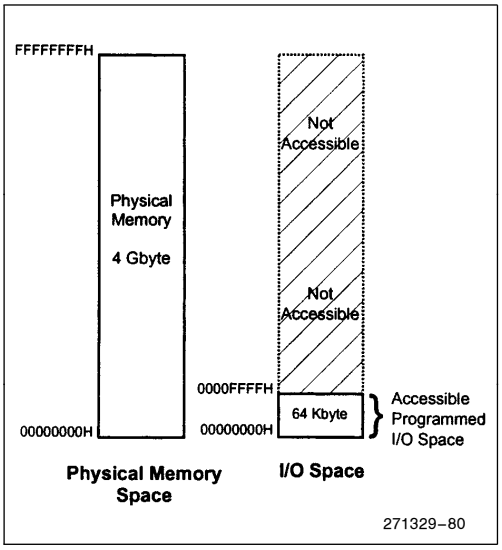


Figure 10-1. Physical Memory and I/O Spaces

10.1.1.1 Memory and I/O Space Organization

The Military Intel486 processor datapath to memory and input/output (I/O) spaces can be 32-, 16- or 8-bits wide. The byte enable signals, BE0# – BE3#, allow byte granularity when addressing any memory or I/O structure whether 8, 16 or 32 bits wide.

The Military Intel486 processor includes bus control pins, BS16# and BS8#, which allow direct connection to 16- and 8-bit memories and I/O devices. Cycles to 32-, 16- and 8-bit may occur in any sequence, since the BS8# and BS16# signals are sampled during each bus cycle.

32-bit wide memory and I/O spaces are organized as arrays of physical 4-byte words. Each memory or I/O 4-byte word has four individually addressable bytes at consecutive byte addresses (see Figure 10-2). The lowest addressed byte is associated with data signals D0–D7; the highest-addressed byte with D24–D31. Physical 4-byte words begin at addresses divisible by four.

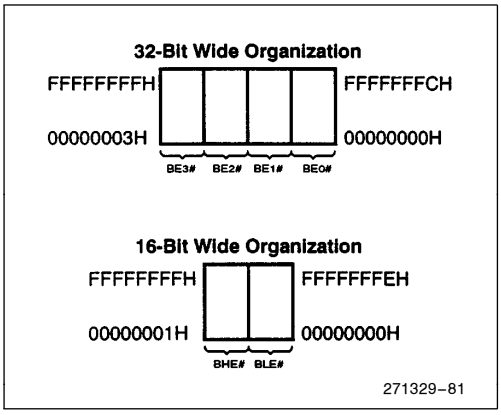


Figure 10-2. Physical Memory and I/O Space Organization

16-bit memories are organized as arrays of physical 2-byte words. Physical 2-byte words begin at addresses divisible by two. The byte enables BE0# – BE3#, must be decoded to A1, BLE# and BHE# to address 16-bit memories. (See section 10.1.3, ''Interfacing with 8-, 16- and 32-Bit Memories.'')

To address 8-bit memories, the two low order address bits A0 and A1, must be decoded from BE0# – BE3#. The same logic can be used for 8- and 16-bit memories, because the decoding logic for BLE# and A0 are the same. (See section 10.1.3, ''Interfacing with 8-, 16-, and 32-Bit Memories.'')

10.1.2 DYNAMIC DATA BUS SIZING

Dynamic data bus sizing is a feature allowing processor connection to 32-, 16- or 8-bit buses for memory or I/O. The Military Intel486 processor may connect to all three bus sizes. Transfers to or from 32-, 16- or 8-bit devices are supported by dynamically determining the bus width during each bus cycle. Address decoding circuitry may assert BS16# for 16-bit devices, or BS8# for 8-bit devices during each bus cycle. BS8# and BS16# must be negated when addressing 32-bit devices. An 8-bit bus width is selected if both BS16# and BS8# are asserted.

BS16# and BS8# force the Military Intel486 processor to run additional bus cycles to complete requests larger than 16- or 8 bits. A 32-bit transfer will be converted into two 16-bit transfers (or 3 transfers if the data is misaligned) when BS16# is asserted. Asserting BS8# will convert a 32-bit transfer into four 8-bit transfers.

Extra cycles forced by BS16# or BS8# should be viewed as independent bus cycles. BS16# or BS8# must be driven active during each of the extra cycles unless the addressed device has the ability to change the number of bytes it can return between cycles.

The Military Intel486 processor will drive the byte enables appropriately during extra cycles forced by BS8# and BS16#. A2–A31 will not change if accesses are to a 32-bit aligned area. Table 10-3 shows the set of byte enables that will be generated on the next cycle for each of the valid possibilities of the byte enables on the current cycle.

The dynamic bus sizing feature of the Military Intel486 processor is significantly different than that of the Intel386 processor. Unlike the Intel386 processor, the Military Intel486 processor requires that data bytes be driven on the addressed data pins. The simplest example of this function is a 32-bit aligned, BS16# read. When the Military Intel486 processor reads the two high order bytes, they must be driven on the data bus pins D16–D31. The Military Intel486 processor expects the two low order bytes on D0–D15. The Intel386 processor expects both the high and low order bytes on D0–D15. The Intel386 processor always reads or writes data on the lower 16 bits of the data bus when BS16# is asserted.

The external system must contain buffers to enable the Military Intel486 processor to read and write data on the appropriate data bus pins. Table 10-4 shows the data bus lines to which the Military Intel486 processor expects data to be returned for each valid combination of byte enables and bus sizing options.

Table 10-3. Next Byte Enable Values for BSn# Cycles

Current				Next with BS8#				Next with BS16#			
BE3#	BE2#	BE1#	BE0#	BE3#	BE2#	BE1#	BE0#	BE3#	BE2#	BE1#	BE0#
1	1	1	0	n	n	n	n	n	n	n	n
1	1	0	0	1	1	0	1	n	n	n	n
1	0	0	0	1	0	0	1	1	0	1	1
0	0	0	0	0	0	0	1	0	0	1	1
1	1	0	1	n	n	n	n	n	n	n	n
1	0	0	1	1	0	1	1	1	0	1	1
0	0	0	1	0	0	1	1	0	0	1	1
1	0	1	1	n	n	n	n	n	n	n	n
0	0	1	1	0	1	1	1	n	n	n	n
0	1	1	1	n	n	n	n	n	n	n	n

NOTE:
''n'' means that another bus cycle will not be required to satisfy the request.

Table 10-4. Data Pins Read with Different Bus Sizes

BE3#	BE2#	BE1#	BE0#	w/o BS8#/BS16#	w BS8#	w BS16#
1	1	1	0	D7–D0	D7–D0	D7–D0
1	1	0	0	D15–D0	D7–D0	D15–D0
1	0	0	0	D23–D0	D7–D0	D15–D0
0	0	0	0	D31–D0	D7–D0	D15–D0
1	1	0	1	D15–D8	D15–D8	D15–D8
1	0	0	1	D23–D8	D15–D8	D15–D8
0	0	0	1	D31–D8	D15–D8	D15–D8
1	0	1	1	D23–D16	D23–D16	D23–D16
0	0	1	1	D31–D16	D23–D16	D31–D16
0	1	1	1	D31–D24	D31–D24	D31–D24

Valid data will only be driven onto data bus pins corresponding to active byte enables during write cycles. Other pins in the data bus will be driven but they will not contain valid data. Unlike the Intel386 processor, the Military Intel486 processor will not duplicate write data onto parts of the data bus for which the corresponding byte enable is negated.

10.1.3 INTERFACING WITH 8-, 16- AND 32-BIT MEMORIES

In 32-bit physical memories, such as the one shown in Figure 10-3, each 4-byte word begins at a byte address that is a multiple of four. A2–A31 are used as a 4-byte word select. BE0# –BE3# select individual bytes within the 4-byte word. BS8# and BS16# are negated for all bus cycles involving the 32-bit array.

16- and 8-bit memories require external byte swapping logic for routing data to the appropriate data lines and logic for generating BHE#, BLE# and A1. In systems where mixed memory widths are used, extra address decoding logic is necessary to assert BS16# or BS8#.

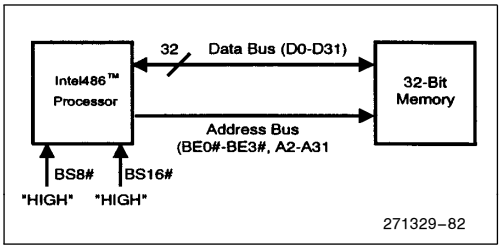


Figure 10-3. Military Intel486™ Processor with 32-Bit Memory

Figure 10-4 shows the Military Intel486 processor address bus interface to 32-, 16- and 8-bit memories. To address 16-bit memories the byte enables must be decoded to produce A1, BHE# and BLE# (A0). For 8-bit wide memories the byte enables must be decoded to produce A0 and A1. The same byte select logic can be used in 16- and 8-bit systems, because BLE# is exactly the same as A0. (See Table 10-5.)

BE0# –BE3# can be decoded as shown in Table 10-5 to generate A1, BHE# and BLE#. The byte select logic necessary to generate BHE# and BLE# is shown in Figure 10-5.

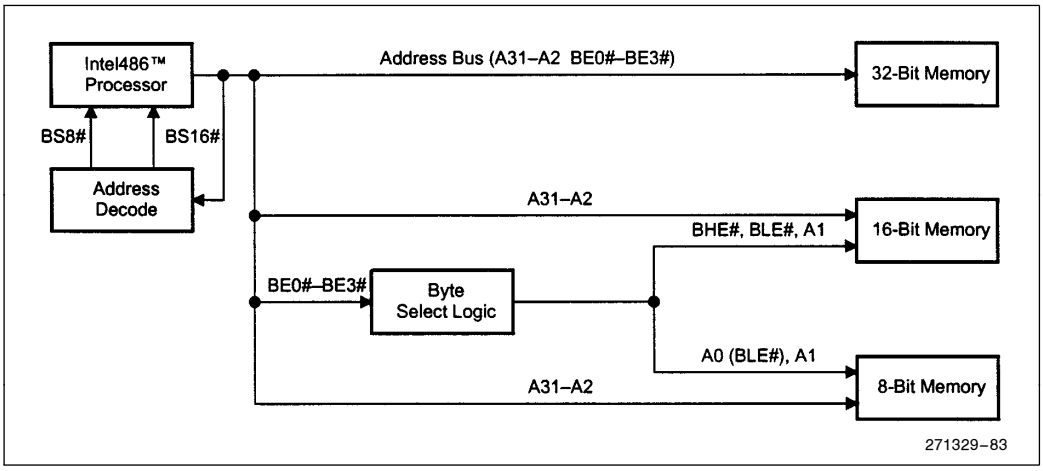


Figure 10-4. Addressing 16- and 8-Bit Memories

Table 10-5. Generating A1, BHE# and BLE# for Addressing 16-Bit Devices

Military Intel486™ Processor				8-, 16-Bit Bus Signals			Comments
BE3#	BE2#	BE1#	BE0#	A1	BHE#	BLE# (A0)	
H*	H*	H*	H*	x	x	x	x–no active bytes
H	H	H	L	L	H	L	
H	H	L	H	L	L	H	
H	H	L	L	L	L	L	
H	L	H	H	H	H	L	
H*	L*	H*	L*	x	x	x	x–not contiguous bytes
H	L	L	H	L	L	H	
H	L	L	L	L	L	L	
L	H	H	H	H	L	H	
L*	H*	H*	L*	x	x	x	x–not contiguous bytes
L*	H*	L*	H*	x	x	x	x–not contiguous bytes
L*	H*	L*	L*	x	x	x	x–not contiguous bytes
L		H	H	H	L	L	
L*	L*	H*	L*	x	x	x	x–not contiguous bytes
L	L	L	H	L	L	H	
L	L	L	L	L	L	L	

BLE# asserted when D0–D7 of 16-bit bus is active.
BHE# asserted when D8–D15 of 16-bit bus is active.
A1 low for all even words; A1 high for all odd words.

Key:
x = don't care H = high voltage level L = low voltage level
* = a non-occurring pattern of Byte Enables; either none are asserted or the pattern has Byte Enables asserted for non-contiguous bytes

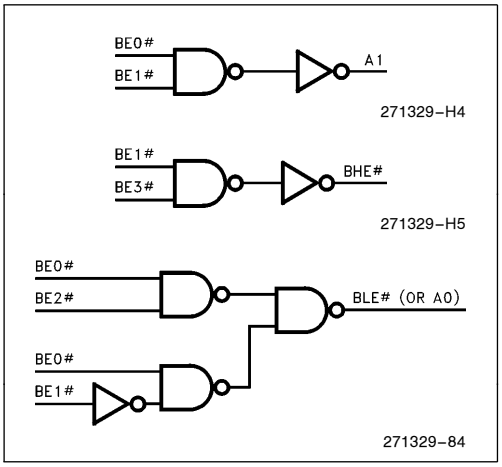


Figure 10-5. Logic to Generate A1, BHE# and BLE# for 16-Bit Buses

Combinations of BE0#–BE3# that never occur are those in which two or three asserted byte enables are separated by one or more negated byte enables. These combinations are "don't care" conditions in the decoder. A decoder can use the non-occurring BE0#–BE3# combinations to its best advantage.

Figure 10-6 shows a Military Intel486 processor data bus interface to 16- and 8-bit wide memories. External byte swapping logic is needed on the data lines so that data is supplied to and received from the Military Intel486 processor on the correct data pins (see Table 10-4).

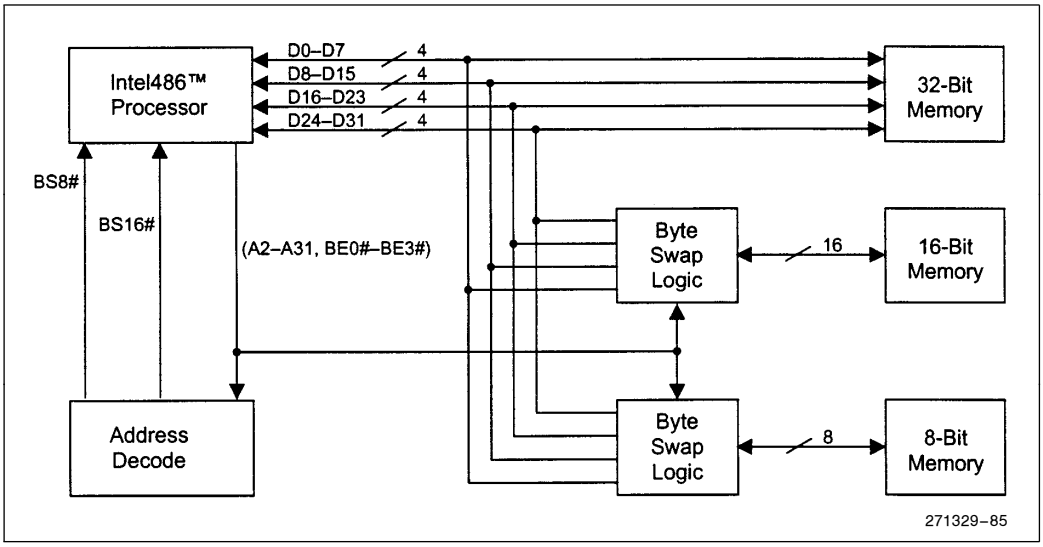


Figure 10-6. Data Bus Interface to 16- and 8-Bit Memories

10.1.4 DYNAMIC BUS SIZING DURING CACHE LINE FILLS

BS8# and BS16# can be driven during cache line fills. The Military Intel486 processor will generate enough 8- or 16-bit cycles to fill the cache line. This can be up to sixteen 8-bit cycles.

The external system should assume that all byte enables are active for the first cycle of a cache line fill. The Military Intel486 processor will generate proper byte enables for subsequent cycles in the line fill. Table 10-6 shows the appropriate A0 (BLE#), A1 and BHE# for the various combinations of the Military Intel486 processor byte enables on both the first and subsequent cycles of the cache line fill. The "*" marks all combinations of byte enables that will be generated by the Military Intel486 processor during a cache line fill.

10.1.5 OPERAND ALIGNMENT

Physical 4-byte words begin at addresses that are multiples of four. It is possible to transfer a logical operand that spans more than one physical 4-byte word of memory or I/O at the expense of extra cycles. Examples are 4-byte operands beginning at addresses that are not evenly divisible by 4, or 2-byte words split between two physical 4-byte words. These are referred to as unaligned transfers.

Operand alignment and data bus size dictate when multiple bus cycles are required. Table 10-7 describes the transfer cycles generated for all combinations of logical operand lengths, alignment, and data bus sizing. When multiple cycles are required to transfer a multibyte logical operand, the highest-order bytes are transferred first. For example, when the processor does a 4-byte unaligned read beginning at location x11 in the 4-byte aligned space, the three high order bytes are read in the first bus cycle. The low byte is read in a subsequent bus cycle.

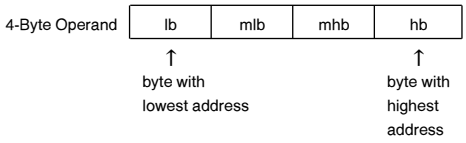
Table 10-6. Generating A0, A1 and BHE# from the Military Intel486™ Processor Byte Enables

				First Cache Fill Cycle			Any Other Cycle		
BE3#	BE2#	BE1#	BE0#	A0	A1	BHE#	A0	A1	BHE#
1	1	1	0	0	0	0	0	0	1
1	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0
*0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	1	0	0
1	0	0	1	0	0	0	1	0	0
*0	0	0	1	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	1
*0	0	1	1	0	0	0	0	1	0
*0	1	1	1	0	0	0	1	1	0

Table 10-7. Transfer Bus Cycles for Bytes, Words and Dwords

	Byte-Length of Logical Operand								
	1	2				4			
Physical Byte Address in Memory (Low Order Bits)	xx	00	01	10	11	00	01	10	11
Transfer Cycles over 32-Bit Bus	b	w	w	w	hb lb	d	hb l3	hw lw	h3 lb
Transfer Cycles over 16-Bit Bus († = BS#16 asserted)	b	w	lb † hb†	w	hb lb	lw † hw†	hb lb† mw†	hw lw	mw† hb† lb
Transfer Cycles over 8-Bit Bus (‡ = BS8# Asserted)	b	lb ‡ hb ‡	lb ‡ hb‡	lb ‡ hb ‡	hb lb	lb ‡ mlb ‡ mhb ‡ hb ‡	hb lb ‡ mlb ‡ mhb ‡	mhb ‡ hb ‡ lb ‡ mlb ‡	mlb ‡ mhb ‡ hb ‡ kb

KEY:
b = byte transfer
w = 2-byte transfer
3 = 3-byte transfer
d = 4-byte transfer
h = high-order portion
l = low-order portion
m = mid-order portion



The function of unaligned transfers with dynamic bus sizing is not obvious. When the external systems asserts BS16# or BS8# forcing extra cycles, low-order bytes or words are transferred first (opposite to the example above). When the Military Intel486 processor requests a 4-byte read and the external system asserts BS16#, the lower 2 bytes are read first followed by the upper 2 bytes.

In the unaligned transfer described above, the processor requested three bytes on the first cycle. If the external system asserted BS16# during this 3-byte transfer, the lower word is transferred first followed by the upper byte. In the final cycle the lower byte of the 4-byte operand is transferred as in the 32-bit example above.

10.2 Bus Functional Description

The Military Intel486 processor supports a wide variety of bus transfers to meet the needs of high performance systems. Bus transfers can be single cycle or multiple cycle, burst or non-burst, cacheable or non-cacheable, 8-, 16- or 32-bit, and pseudo-locked.

To support multiprocessing systems there are cache invalidation cycles and locked cycles.

This section begins with basic non-cacheable non-burst single cycle transfers. It moves on to multiple cycle transfers and introduces the burst mode. Cacheability is introduced in section 10.2.3, ''Cacheable Cycles.'' The remaining sections describe locked, pseudo-locked, invalidate, bus hold and interrupt cycles.

Bus cycles and data cycles are discussed in this section. A bus cycle is at least two clocks long and begins with ADS# active in the first clock and ready active in the last clock. Data is transferred to or from the Military Intel486 processor during a data cycle. A bus cycle contains one or more data cycles.

Refer to section 10.2.13, ''Bus States,'' for a description of the bus states shown in the timing diagrams.

10.2.1 NON-CACHEABLE NON-BURST SINGLE CYCLE

10.2.1.1 No Wait States

The fastest non-burst bus cycle that the Military Intel486 processor supports is two clocks long. These cycles are called 2-2 cycles because reads and writes take two cycles each. The first ''2'' refers to reads and the second to writes.

For example, if a wait state needs to be added to the write, the cycle would be called 2-3.

Basic two clock read and write cycles are shown in Figure 10-7. The Military Intel486 processor initiates a cycle by asserting the address status signal (ADS#) at the rising edge of the first clock. The ADS# output indicates that a valid bus cycle definition and address is available on the cycle definition lines and address bus.

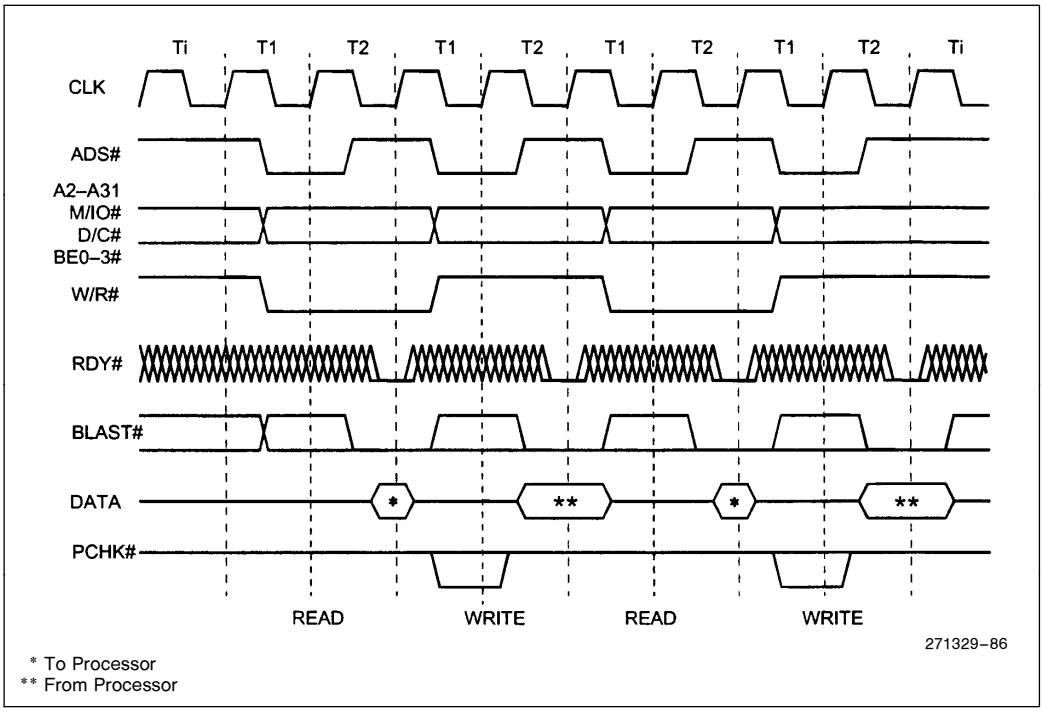


Figure 10-7. Basic 2-2 Bus Cycle

The non-burst ready input (RDY#) is returned by the external system in the second clock. RDY# indicates that the external system has presented valid data on the data pins in response to a read or the external system has accepted data in response to a write.

The Military Intel486 processor samples RDY# at the end of the second clock. The cycle is complete if RDY# is active (LOW) when sampled. Note that RDY# is ignored at the end of the first clock of the bus cycle.

The burst last signal (BLAST#) is asserted (LOW) by the Military Intel486 processor during the second clock of the first cycle in all bus transfers illustrated in Figure 10-7. This indicates that each transfer is complete after a single cycle. The Military Intel486 processor asserts BLAST# in the last cycle of a bus transfer.

The timing of the parity check output (PCHK#) is shown in Figure 10-7. The Military Intel486 processor drives the PCHK# output one clock after ready terminates a read cycle. PCHK# indicates the parity status for the data sampled at the end of the previous clock. The PCHK# signal can be used by the external system. The Military Intel486 processor does nothing in response to the PCHK# output.

10.2.1.2 Inserting Wait States

The external system can insert wait states into the basic 2-2 cycle by driving RDY# inactive at the end of the second clock. RDY# must be driven inactive to insert a wait state. Figure 10-8 illustrates a simple non-burst, non-cacheable signal with one wait state added. Any number of wait states can be added to a Military Intel486 processor bus cycle by maintaining RDY# inactive.

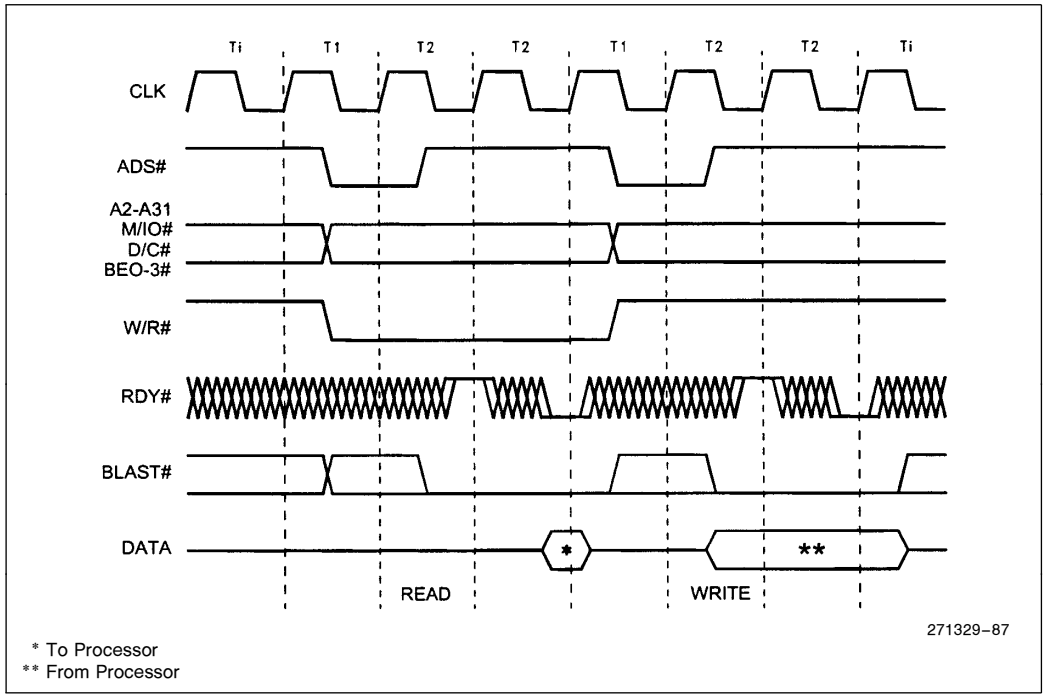


Figure 10-8. Basic 3-3 Bus Cycle

The burst ready input (BRDY#) must be driven inactive on all clock edges where RDY# is driven inactive for proper operation of these simple non-burst cycles.

10.2.2 MULTIPLE AND BURST CYCLE BUS TRANSFERS

Multiple cycle bus transfers can be caused by internal requests from the Military Intel486 processor or by the external memory system. An internal request for a 128-bit pre-fetch must take more than one cycle. Internal requests for unaligned data may also require multiple bus cycles. A cache line fill requires multiple cycles to complete.

The external system can cause a multiple cycle transfer when it can only supply 8- or 16-bits per cycle.

Only multiple cycle transfers caused by internal requests are considered in this section. Cacheable cycles and 8- and 16-bit transfers are covered in section 10.2.3, ''Cacheable Cycles'' and section 10.2.5, ''8- and 16-Bit Cycles.''

Internal Requests from Military Intel486 DX, IntelDX2, and IntelDX4 Processors

An internal request by a Military Intel486 DX, IntelDX2, or IntelDX4 processor for a 64-bit floating point load must take more than one internal cycle.

10.2.2.1 Burst Cycles

The Military Intel486 processor can accept burst cycles for any bus requests that require more than a single data cycle. During burst cycles, a new data item is strobed into the Military Intel486 processor every clock rather than every other clock as in non-burst cycles. The fastest burst cycle requires 2 clocks for the first data item with subsequent data items returned every clock.

The Military Intel486 processor is capable of bursting a maximum of 32 bits during a write. Burst writes can only occur if BS8# or BS16# is asserted. For example, the Military Intel486 processor can burst write four 8-bit operands or two 16-bit operands in a single burst cycle. But the Military Intel486 processor cannot burst multiple 32-bit writes in a single burst cycle.

Burst cycles begin with the Military Intel486 processor driving out an address and asserting ADS# in the same manner as non-burst cycles. The Military Intel486 processor indicates that it is willing to perform a burst cycle by holding the burst last signal (BLAST#) inactive in the second clock of the cycle. The external system indicates its willingness to do a burst cycle by returning the burst ready signal (BRDY#) active.

The addresses of the data items in a burst cycle will all fall within the same 16-byte aligned area (corresponding to an internal Military Intel486 processor cache line). A 16-byte aligned area begins at location XXXXXXX0 and ends at location XXXXXXXF. During a burst cycle, only BE0–3#, A_2, and A_3 may change. A_4–A_{31}, M/IO#, D/C#, and W/R# will remain stable throughout a burst. Given the first address in a burst, external hardware can easily calculate the address of subsequent transfers in advance. An external memory system can be designed to quickly fill the Military Intel486 processor internal cache lines.

Burst cycles are not limited to cache line fills. Any multiple cycle read request by the Military Intel486 processor can be converted into a burst cycle. The Military Intel486 processor will only burst the number of bytes needed to complete a transfer.

For example, the Military Intel486 DX, IntelDX2, Write-Back Enhanced IntelDX2 or IntelDX4 processor will burst eight bytes for a 64-bit floating point non-cacheable read.

The external system converts a multiple cycle request into a burst cycle by returning BRDY# active rather than RDY# (non-burst ready) in the first cycle of a transfer. For cycles that cannot be burst, such as interrupt acknowledge and halt, BRDY# has the same effect as RDY#. BRDY# is ignored if both BRDY# and RDY# are returned in the same clock. Memory areas and peripheral devices that cannot perform bursting must terminate cycles with RDY#.

10.2.2.2 Terminating Multiple and Burst Cycle Transfers

The Military Intel486 processor drives BLAST# inactive for all but the last cycle in a multiple cycle transfer. BLAST# is driven inactive in the first cycle to inform the external system that the transfer could take additional cycles. BLAST# is driven active in the last cycle of the transfer indicating that the next time BRDY# or RDY# is returned the transfer is complete.

BLAST# is not valid in the first clock of a bus cycle. It should be sampled only in the second and subsequent clocks when RDY# or BRDY# is returned.

The number of cycles in a transfer is a function of several factors including the number of bytes the Military Intel486 processor needs to complete an internal request (1, 2, 4, 8, or 16), the state of the bus size inputs (BS8# and BS16#), the state of the cache enable input (KEN#) and alignment of the data to be transferred.

When the Military Intel486 processor initiates a request it knows how many bytes will be transferred and if the data is aligned. The external system must indicate whether the data is cacheable (if the transfer is a read) and the width of the bus by returning the state of the KEN#, BS8# and BS16# inputs one clock before RDY# or BRDY# is returned. The Military Intel486 processor determines how many cycles a transfer will take based on its internal information and inputs from the external system.

BLAST# is not valid in the first clock of a bus cycle because the Military Intel486 processor cannot determine the number of cycles a transfer will take until the external system returns KEN#, BS8# and BS16#. BLAST# should only be sampled in the second and subsequent clocks of a cycle when the external system returns RDY# or BRDY#.

The system may terminate a burst cycle by returning RDY# instead of BRDY#. BLAST# will remain deasserted until the last transfer. However, any transfers required to complete a cache line fill will follow the burst order, e.g., if burst order was 4, 0, C, 8 and RDY# was returned at after 0, the next transfers will be from C and 8.

10.2.2.3 Non-Cacheable, Non-Burst, Multiple Cycle Transfers

Figure 10-9 illustrates a 2 cycle non-burst, non-cacheable multiple cycle read. This transfer is simply a sequence of two single cycle transfers. The Military Intel486 processor indicates to the external system that this is a multiple cycle transfer by driving BLAST# inactive during the second clock of the first cycle. The external system returns RDY# active indicating that it will not burst the data. The external system also indicates that the data is not cacheable by returning KEN# inactive one clock before it returns RDY# active. When the Military Intel486 processor samples RDY# active it ignores BRDY#.

Each cycle in the transfer begins when ADS# is driven active and the cycle is complete when the external system returns RDY# active.

The Military Intel486 processor indicates the last cycle of the transfer by driving BLAST# active. The next RDY# returned by the external system terminates the transfer.

10.2.2.4 Non-Cacheable Burst Cycles

The external system converts a multiple cycle request into a burst cycle by returning BRDY# active rather than RDY# in the first cycle of the transfer. This is illustrated in Figure 10-10.

There are several features to note in the burst read. ADS# is only driven active during the first cycle of the transfer. RDY# must be driven inactive when BRDY# is returned active.

BLAST# behaves exactly as it does in the non-burst read. BLAST# is driven inactive in the second clock of the first cycle of the transfer indicating more cycles to follow. In the last cycle, BLAST# is driven active telling the external memory system to end the burst after returning the next BRDY#.

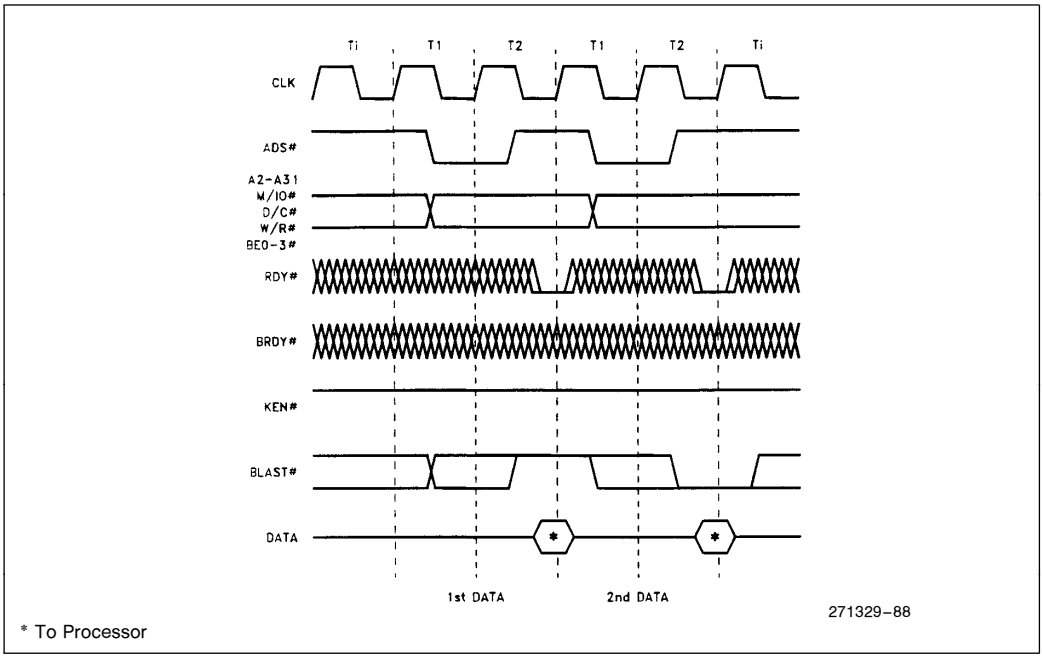


Figure 10-9. Non-Cacheable, Non-Burst, Multiple-Cycle Transfers

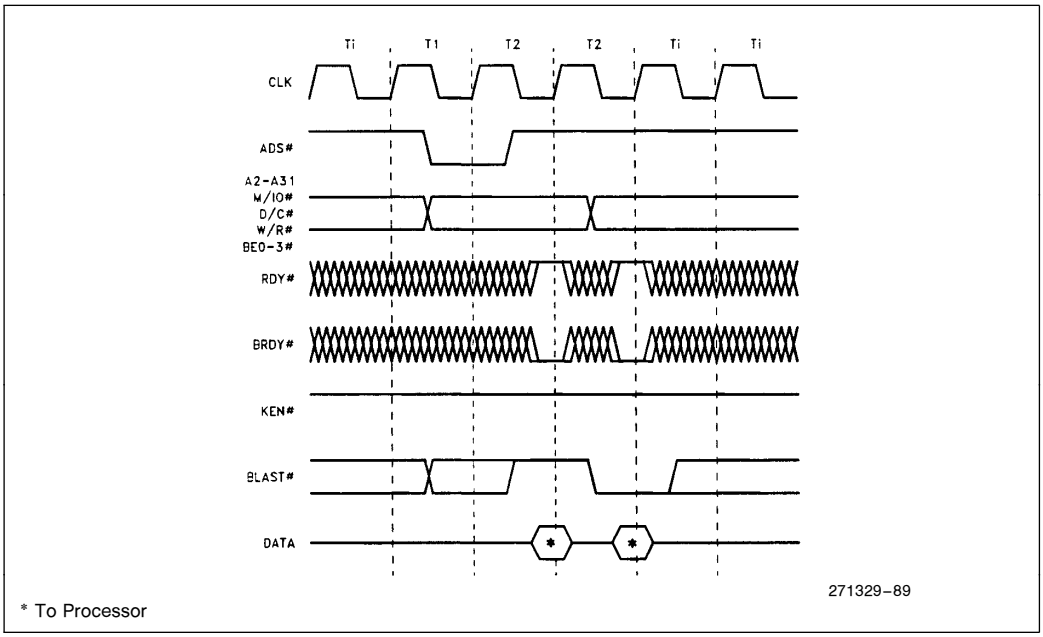


Figure 10-10. Non-Cacheable Burst Cycle

10.2.3 CACHEABLE CYCLES

Any memory read can become a cache fill operation. The external memory system can allow a read request to fill a cache line by returning KEN# active one clock before RDY# or BRDY# during the first cycle of the transfer on the external bus. Once KEN# is asserted and the remaining three requirements described below are met, the Military Intel486 processor will fetch an entire cache line regardless of the state of KEN#. KEN# must be returned active in the last cycle of the transfer for the data to be written into the internal cache. The Military Intel486 processor will only convert memory reads or prefetches into a cache fill.

KEN# is ignored during write or I/O cycles. Memory writes will only be stored in the on-chip cache if there is a cache hit. I/O space is never cached in the internal cache.

To transform a read or a prefetch into a cache line fill the following conditions must be met:

1. The KEN# pin must be asserted one clock prior to RDY# or BRDY# being returned for the first data cycle.
2. The cycle must be of the type that can be internally cached. (Locked reads, I/O reads, and interrupt acknowledge cycles are never cached).
3. The page table entry must have the page cache disable bit (PCD) set to 0. To cache a page table entry, the page directory must have PCD = 0. To cache reads or prefetches when paging is disabled, or to cache the page directory entry, control register 3 (CR3) must have PCD = 0.
4. The cache disable (CD) bit in control register 0 (CR0) must be clear.

External hardware can determine when the Military Intel486 processor has transformed a read or prefetch into a cache fill by examining the KEN#, M/IO#, D/C#, W/R#, LOCK#, and PCD pins. These pins convey to the system the outcome of conditions 1–3 in the above list. In addition, the Military Intel486 processor drives PCD high whenever the CD bit in CR0 is set, so that external hardware can evaluate condition 4.

Cacheable cycles can be burst or non-burst.

10.2.3.1 Byte Enables during a Cache Line Fill

For the first cycle in the line fill, the state of the byte enables should be ignored. In a non-cacheable memory read, the byte enables indicate the bytes actually required by the memory or code fetch.

The Military Intel486 processor expects to receive valid data on its entire bus (32 bits) in the first cycle of a cache line fill. Data should be returned with the assumption that all the byte enable pins are driven active. However if BS8# is asserted only one byte need be returned on data lines D0–D7. Similarly if BS16# is asserted two bytes should be returned on D0–D15.

The Military Intel486 processor will generate the addresses and byte enables for all subsequent cycles in the line fill. The order in which data is read during a line fill depends on the address of the first item read. Byte ordering is discussed in section 10.2.4, ‘‘Burst Mode Details.’’

10.2.3.2 Non-Burst Cacheable Cycles

Figure 10-11 shows a non-burst cacheable cycle. The cycle becomes a cache fill when the Military Intel486 processor samples KEN# active at the end of the first clock. The Military Intel486 processor drives BLAST# inactive in the second clock in response to KEN#. BLAST# is driven inactive because a cache fill requires 3 additional cycles to complete. BLAST# remains inactive until the last transfer in the cache line fill. KEN# must be returned active in the last cycle of the transfer for the data to be written into the internal cache.

Note that this cycle would be a single bus cycle if KEN# was not sampled active at the end of the first clock. The subsequent three reads would not have happened since a cache fill was not requested.

The BLAST# output is invalid in the first clock of a cycle. BLAST# may be active during the first clock due to earlier inputs. Ignore BLAST# until the second clock.

During the first cycle of the cache line fill the external system should treat the byte enables as if they are all active. In subsequent cycles in the burst, the Military Intel486 processor drives the address lines and byte enables. (See section 10.2.4.2, "Burst and Cache Line Fill Order") .

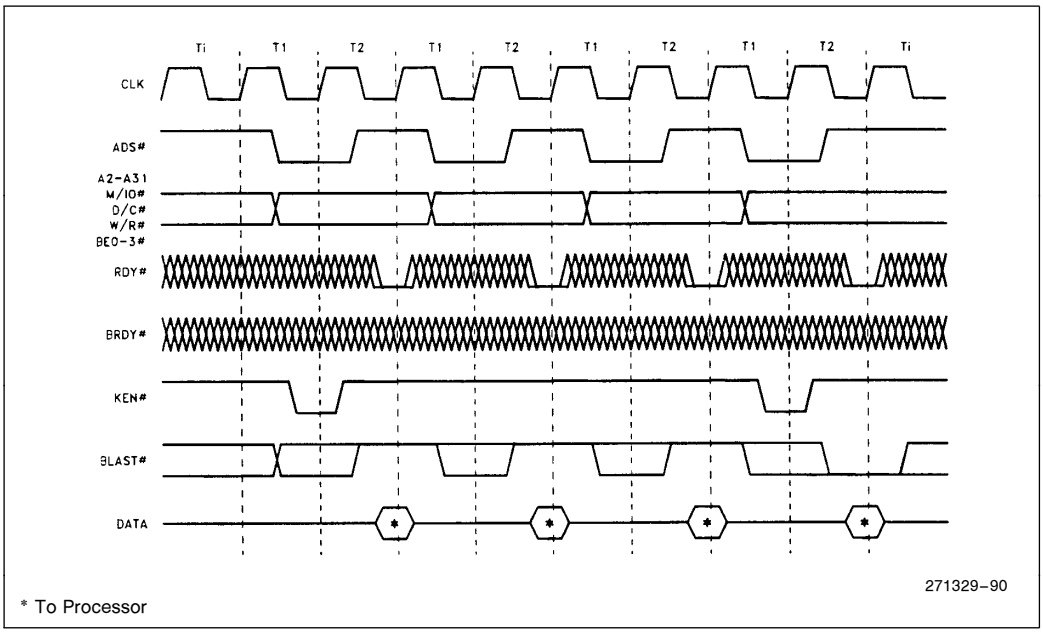


Figure 10-11. Non-Burst, Cacheable Cycles

10.2.3.3 Burst Cacheable Cycles

Figure 10-12 illustrates a burst mode cache fill. As in Figure 10-11, the transfer becomes a cache line fill when the external system returns KEN# active at the end of the first clock in the cycle.

The external system informs the Military Intel486 processor that it will burst the line in by driving BRDY# active at the end of the first cycle in the transfer.

Note that during a burst cycle, ADS# is only driven with the first address.

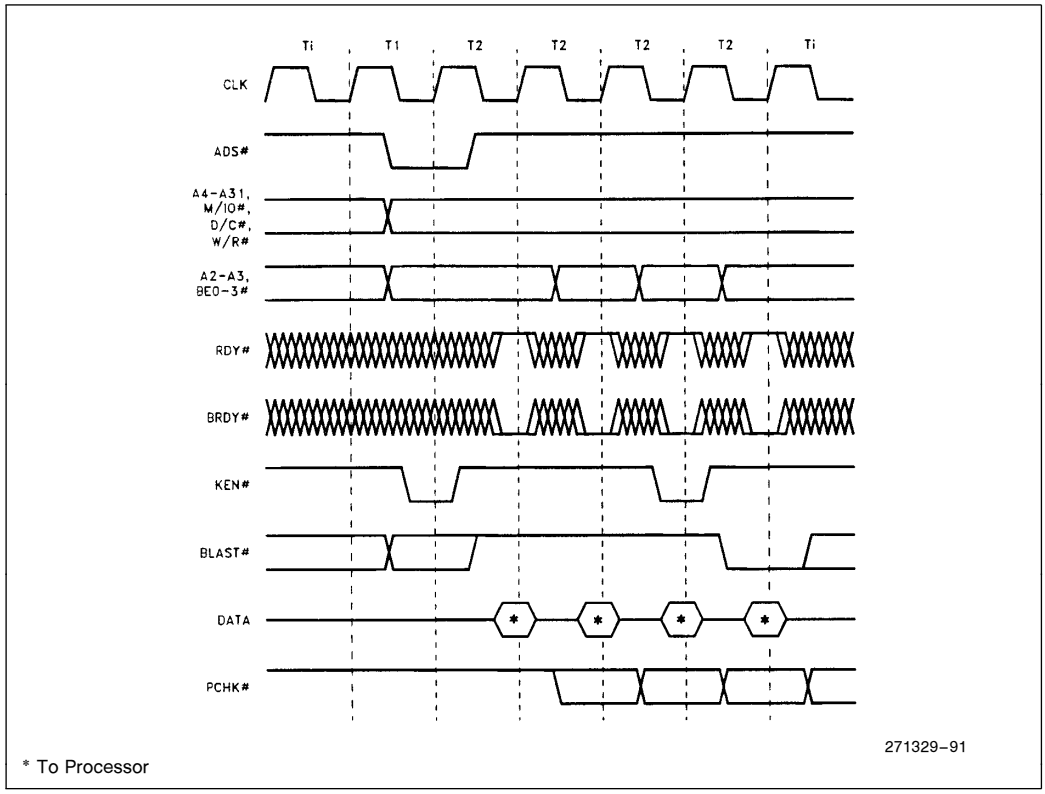


Figure 10-12. Burst Cacheable Cycle

10.2.3.4 Effect of Changing KEN# during a Cache Line Fill

KEN# can change multiple times as long as it arrives at its final value in the clock before RDY# or BRDY# is returned. This is illustrated in Figure 10-13. Note that the timing of BLAST# follows that of KEN# by one clock. The Military Intel486 processor samples KEN# every clock and uses the value returned in the clock before ready to determine if a bus cycle would be a cache line fill. Similarly, it uses the value of KEN# in the last cycle before early RDY# to load the line just retrieved from memory into the cache. KEN# is sampled every clock and it must satisfy setup and hold time.

KEN# can also change multiple times before a burst cycle, as long as it arrives at its final value one clock before ready is returned active.

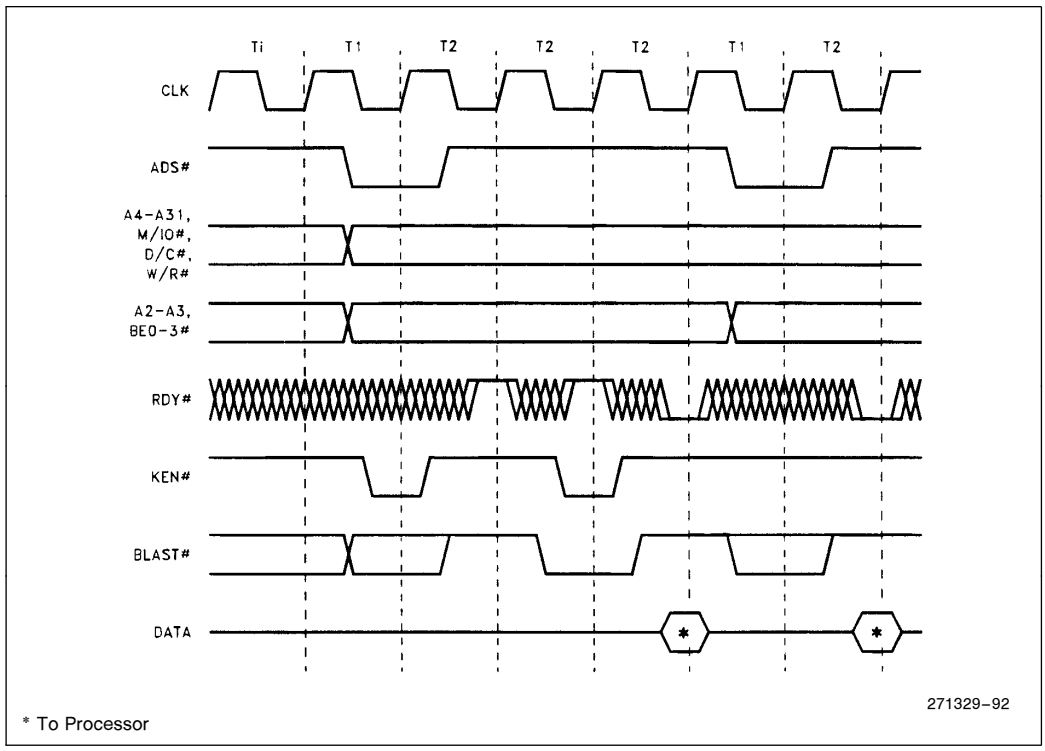


Figure 10-13. Effect of Changing KEN#

10.2.4 BURST MODE DETAILS

10.2.4.1 Adding Wait States to Burst Cycles

Burst cycles need not return data on every clock. The Military Intel486 processor will only strobe data into the chip when either RDY# or BRDY# are active. Driving BRDY# and RDY# inactive adds a wait state to the transfer. A burst cycle where two clocks are required for every burst item is shown in Figure 10-14.

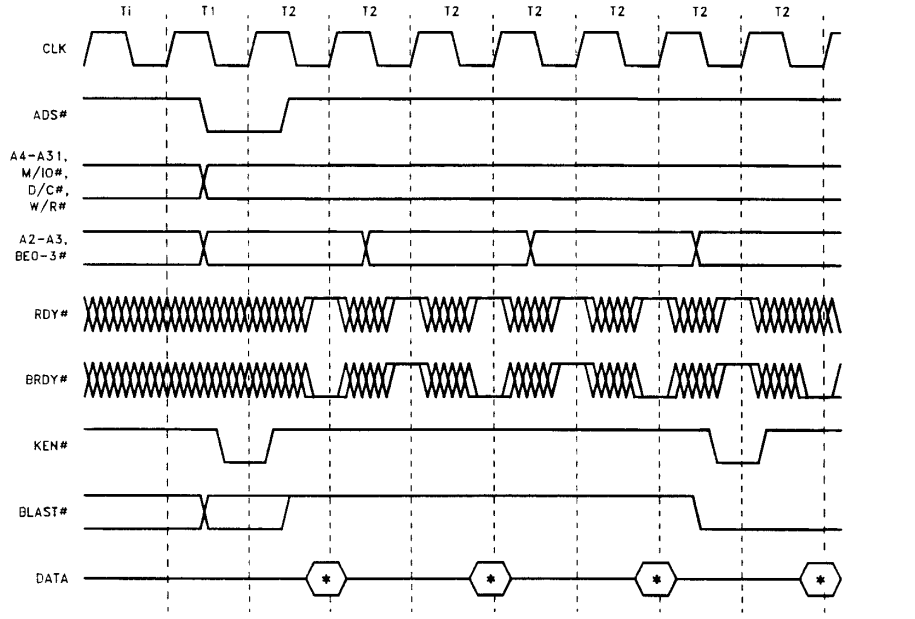


Figure 10-14. Slow Burst Cycle

10.2.4.2 Burst and Cache Line Fill Order

The burst order used by the Military Intel486 processor is shown in Table 10-8. This burst order is followed by any burst cycle (cache or not), cache line fill (burst or not) or code prefetch.

The Military Intel486 processor presents each request for data in an order determined by the first address in the transfer. For example, if the first address was 104 the next three addresses in the burst will be 100, 10C and 108. An example of burst address sequencing is shown in Figure 10-15.

Table 10-8. Burst Order (Both Read and Write Bursts)

First Addr.	Second Addr.	Third Addr.	Fourth Addr.
0	4	8	C
4	0	C	8
8	C	0	4
C	8	4	0

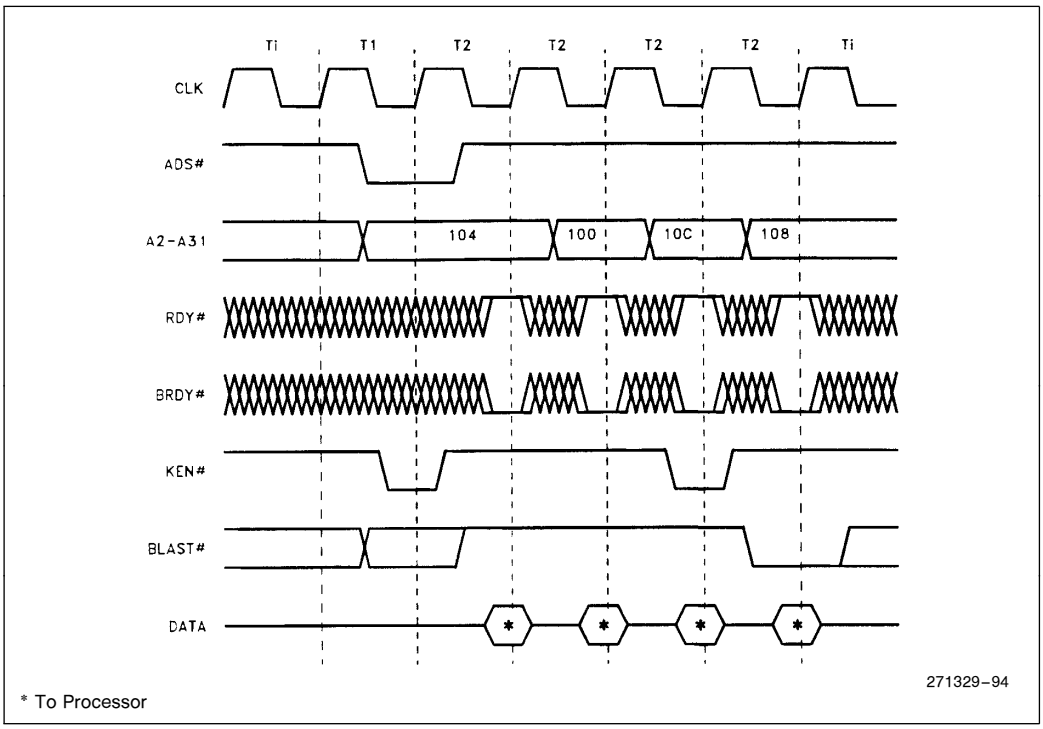


Figure 10-15. Burst Cycle Showing Order of Addresses

The sequences shown in Table 10-8 accommodate systems with 64-bit buses as well as systems with 32-bit data buses. The sequence applies to all bursts, regardless of whether the purpose of the burst is to fill a cache line, do a 64-bit read, or do a pre-fetch. If either BS8# or BS16# is returned active, the Military Intel486 processor completes the transfer of the current 32-bit word before progressing to the next 32-bit word. For example, a BS16# burst to address 4 has the following order: 4-6-0-2-C-E-8-A.

10.2.4.3 Interrupted Burst Cycles

Some memory systems may not be able to respond with burst cycles in the order defined in Table 10-8. To support these systems the Military Intel486 processor allows a burst cycle to be interrupted at any time. The Military Intel486 processor will automatically generate another normal bus cycle after being interrupted to complete the data transfer. This is called an interrupted burst cycle. The external system can respond to an interrupted burst cycle with another burst cycle.

The external system can interrupt a burst cycle by returning RDY# instead of BRDY#. RDY# can be returned after any number of data cycles terminated with BRDY#.

An example of an interrupted burst cycle is shown in Figure 10-16. The Military Intel486 processor immediately drives ADS# active to initiate a new bus cycle after RDY# is returned active. BLAST# driven inactive one clock after ADS# begins the second bus cycle indicating that the transfer is not complete.

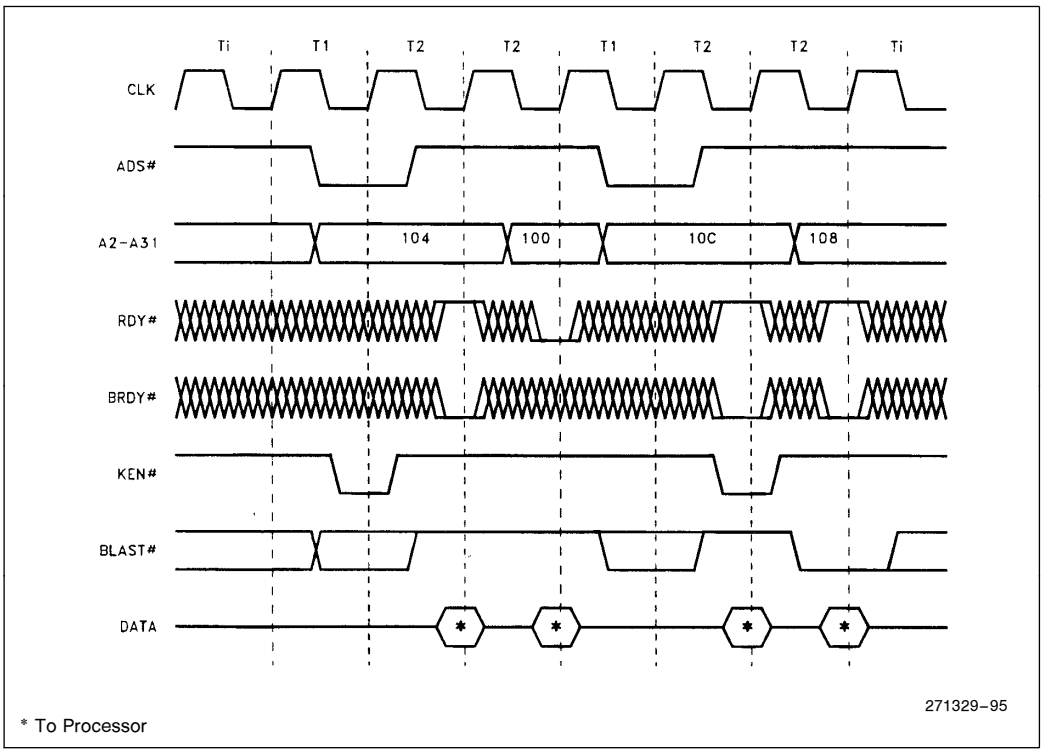


Figure 10-16. Interrupted Burst Cycle

KEN# need not be returned active in the first data cycle of the second part of the transfer in Figure 10-16. The cycle had been converted to a cache fill in the first part of the transfer and the Military Intel486 processor expects the cache fill to be completed. Note that the first half and second half of the transfer in Figure 10-16 are each two cycle burst transfers.

The order in which the Military Intel486 processor requests operands during an interrupted burst transfer is determined by Table 10-7. Mixing RDY# and BRDY# does not change the order in which operand addresses are requested by the Military Intel486 processor.

An example of the order in which the Military Intel486 processor requests operands during a cycle in which the external system mixes RDY# and BRDY# is shown in Figure 10-17. The Military Intel486 processor initially requests a transfer beginning at location 104. The transfer becomes a cache line fill when the external system returns KEN# active. The first cycle of the cache fill transfers the contents of location 104 and is terminated with RDY#. The Military Intel486 processor drives out a new request (by asserting ADS#) to address 100. If the external system terminates the second cycle with BRDY#, the Military Intel486 processor will next request/expect address 10C. The correct order is determined by the first cycle in the transfer, which may not be the first cycle in the burst if the system mixes RDY# with BRDY#.

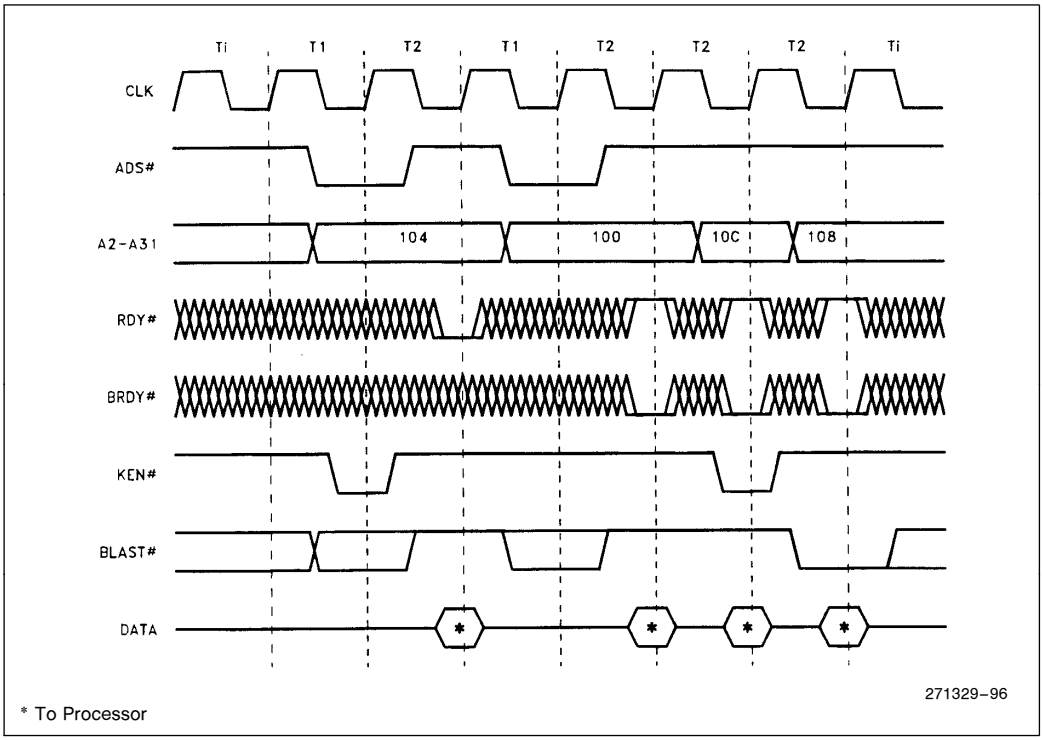


Figure 10-17. Interrupted Burst Cycle with Unobvious Order of Addresses

10.2.5 8- AND 16-BIT CYCLES

The Military Intel486 processor supports both 16- and 8-bit external buses through the BS16# and BS8# inputs. BS16# and BS8# allow the external system to specify, on a cycle by cycle basis, whether the addressed component can supply 8, 16 or 32 bits. BS16# and BS8# can be used in burst cycles as well as non-burst cycles. If both BS16# and BS8# are returned active for any bus cycle, the Military Intel486 processor will respond as if only BS8# were active.

The timing of BS16# and BS8# is the same as that of KEN#. BS16# and BS8# must be driven active before the first RDY# or BRDY# is driven active. Driving the BS16# and BS8# active can force the Military Intel486 processor to run additional cycles to complete what would have been only a single 32-bit cycle. BS8# and BS16# may change the state of BLAST# when they force subsequent cycles from the transfer.

Figure 10-18. shows an example in which BS8# forces the Military Intel486 processor to run two extra cycles to complete a transfer. The Military Intel486 processor issues a request for 24 bits of information. The external system drives BS8# active indicating that only eight bits of data can be supplied per cycle. The Military Intel486 processor issues two extra cycles to complete the transfer.

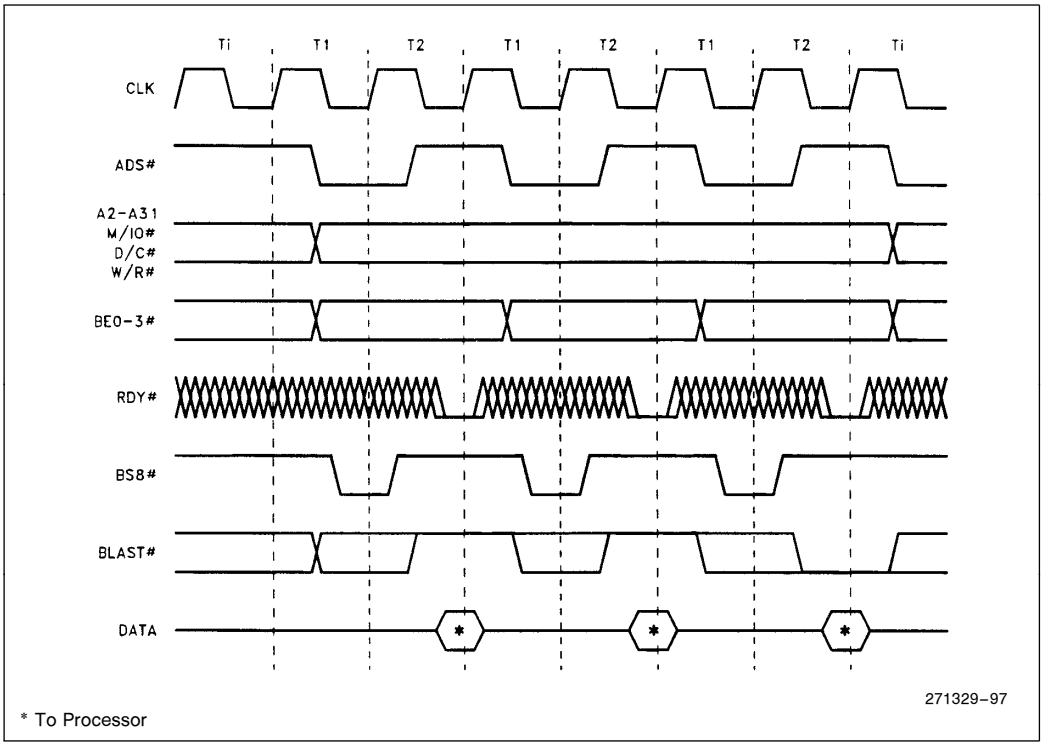


Figure 10-18. 8-Bit Bus Size Cycle

Extra cycles forced by the BS16# and BS8# should be viewed as independent bus cycles. BS16# and BS8# should be driven active for each additional cycle unless the addressed device has the ability to change the number of bytes it can return between cycles. The Military Intel486 processor will drive BLAST# inactive until the last cycle before the transfer is complete.

Refer to section 10.1.2, ''Dynamic Data Bus Sizing,'' for the sequencing of addresses while BS8# or BS16# are active.

BS8# and BS16# operate during burst cycles in exactly the same manner as non-burst cycles. For example, a single non-cacheable read could be transferred by the Military Intel486 processor as four 8-bit burst data cycles. Similarly, a single 32-bit write could be written as four 8-bit burst data cycles. An example of a burst write is shown in Figure 10-19. Burst writes can only occur if BS8# or BS16# is asserted.

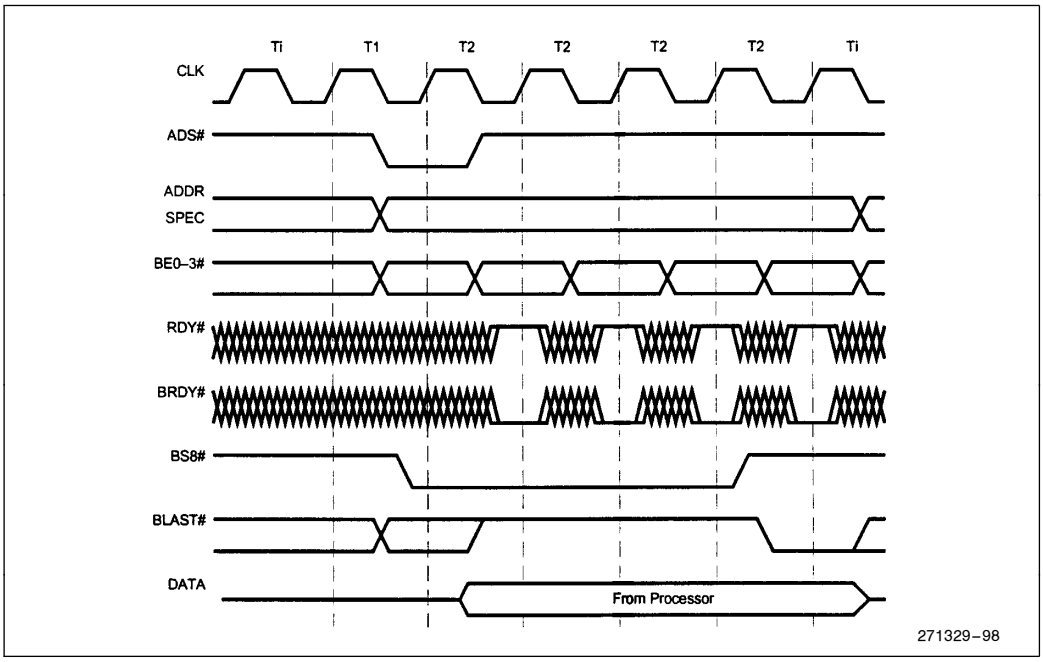


Figure 10-19. Burst Write as a Result of BS8# or BS16#

10.2.6 LOCKED CYCLES

Locked cycles are generated in software for any instruction that performs a read-modify-write operation. During a read-modify-write operation the Military Intel486 processor can read and modify a variable in external memory and be assured that the variable is not accessed between the read and write.

Locked cycles are automatically generated during certain bus transfers. The XCHG (exchange) instruction generates a locked cycle when one of its operands is memory based. Locked cycles are generated when a segment or page table entry is updated and during interrupt acknowledge cycles. Locked cycles are also generated when the LOCK instruction prefix is used with selected instructions.

Locked cycles are implemented in hardware with the LOCK# pin. When LOCK# is active, the Military Intel486 processor is performing a read-modify-write operation and the external bus should not be relinquished until the cycle is complete. Multiple reads or writes can be locked. A locked cycle is shown in Figure 10-20. LOCK# goes active with the address and bus definition pins at the beginning of the first read cycle and remains active until RDY# is returned for the last write cycle. For unaligned 32 bits read-modify-write operation, the LOCK# remains active for the entire duration of the multiple cycle. It will go inactive when RDY# is returned for the last write cycle.

When LOCK# is active, the Military Intel486 processor will recognize address hold and backoff but will not recognize bus hold. It is left to the external system to properly arbitrate a central bus when the Military Intel486 processor generates LOCK#.

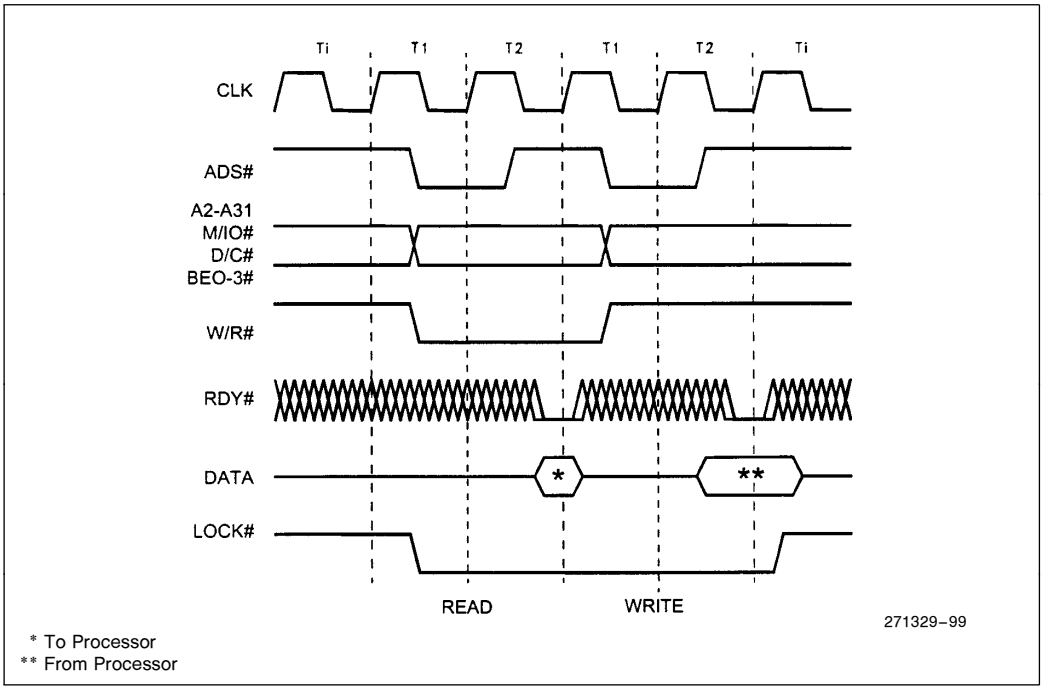


Figure 10-20. Locked Bus Cycle

10.2.7 PSEUDO-LOCKED CYCLES

Pseudo-locked cycles assure that no other master will be given control of the bus during operand transfers which take more than one bus cycle.

For the Military Intel486 processor, examples include 64-bit description loads and cache line fills.

Pseudo-locked transfers are indicated by the PLOCK# pin. The memory operands must be aligned for correct operation of a pseudo-locked cycle.

PLOCK# need not be examined during burst reads. A 64-bit aligned operand can be retrieved in one burst (note: this is only valid in systems that do not interrupt bursts).

The system must examine PLOCK# during 64-bit writes since the Military Intel486 processor cannot burst write more than 32 bits. However, burst can be used within each 32-bit write cycle if BS8# or BS16# is asserted. BLAST will be de-asserted in response to BS8# or BS16#. A 64-bit write will be driven out as two non-burst bus cycles. BLAST# is asserted during both writes since a burst is not possible. PLOCK# is asserted during the first write to indicate that another write follows. This behavior is shown in Figure 10-21.

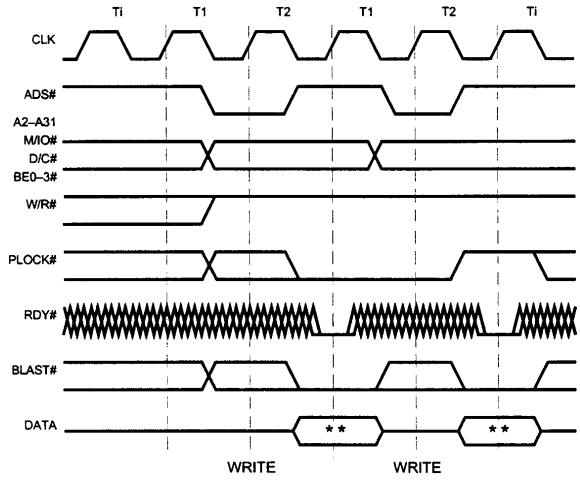
The first cycle of a 64-bit floating point write is the only case in which both PLOCK# and BLAST# are asserted. Normally PLOCK# and BLAST# are the inverse of each other.

During all of the cycles where PLOCK# is asserted, HOLD is not acknowledged until the cycle completes. This results in a large HOLD latency, especially when BS8# or BS16# is asserted. To reduce the HOLD latency during these cycles, windows are available between transfers to allow HOLD to be acknowledged during non-cacheable code prefetches. PLOCK# will be asserted since BLAST# is negated, but it is ignored and HOLD is recognized during the prefetch.

PLOCK# can change several times during a cycle settling to its final value in the clock ready is returned.

10.2.7.1 Floating Point Read and Write Cycles

For Military Intel486 DX, IntelDX2, Write-Back Enhanced IntelDX2, and IntelDX4 processors, 64-bit floating point read and write cycles are also examples of operand transfers that take more than one bus cycle.



** From Processor

Figure 10-21. Pseudo Lock Timing

10.2.8 INVALIDATE CYCLES

Invalidate cycles are needed to keep the Military Intel486 processor internal cache contents consistent with external memory. The Military Intel486 processor contains a mechanism for listening to writes by other devices to external memory. When the Military Intel486 processor finds a write to a section of external memory contained in its internal cache, the Military Intel486 processor's internal copy is invalidated.

Invalidations use two pins, address hold request (AHOLD) and valid external address (EADS#). There are two steps in an invalidation cycle. First, the external system asserts the AHOLD input forcing the Military Intel486 processor to immediately relinquish its address bus. Next, the external system asserts EADS# indicating that a valid address is on the Military Intel486 processor address bus. Figure 10-22 shows the fastest possible invalidation cycle. The Military Intel486 processor recognizes AHOLD on one CLK edge and floats the address bus in response. To allow the address bus to float and avoid contention, EADS# and the invalidation address should not be driven until the following CLK edge. The Military Intel486 processor reads the address over its address lines. If the Military Intel486 processor finds this address in its internal cache, the cache entry is invalidated. Note that the Military Intel486 processor address bus is input/output, unlike the Intel386 processor's bus, which is output only.

The Military Intel486 processor immediately relinquishes its address bus in the next clock upon assertion of AHOLD. For example, the bus could be 3 wait states into a read cycle. If AHOLD is activated, the Military Intel486 processor will immediately float its address bus before ready is returned terminating the bus cycle.

When AHOLD is asserted only the address bus is floated, the data bus can remain active. Data can be returned for a previously specified bus cycle during address hold. (See Figure 10-22 and Figure 10-23.)

EADS# is normally asserted when an external master drives an address onto the bus. AHOLD need not be driven for EADS# to generate an internal invalidate. If EADS# alone is asserted while the Military Intel486 processor is driving the address bus, it is possible that the invalidation address will come from the Military Intel486 processor itself.

Note that it is also possible to run an invalidation cycle by asserting EADS# when BOFF# is asserted or after HLDA has been returned, following the assertion of HOLD.

Running an invalidate cycle prevents the Military Intel486 processor cache from satisfying other internal requests, so invalidations should be run only when necessary. The fastest possible invalidate cycle is shown in Figure 10-22, while a more realistic invalidation cycle is shown in Figure 10-23. Both of the examples take one clock of cache access from the Military Intel486 processor.

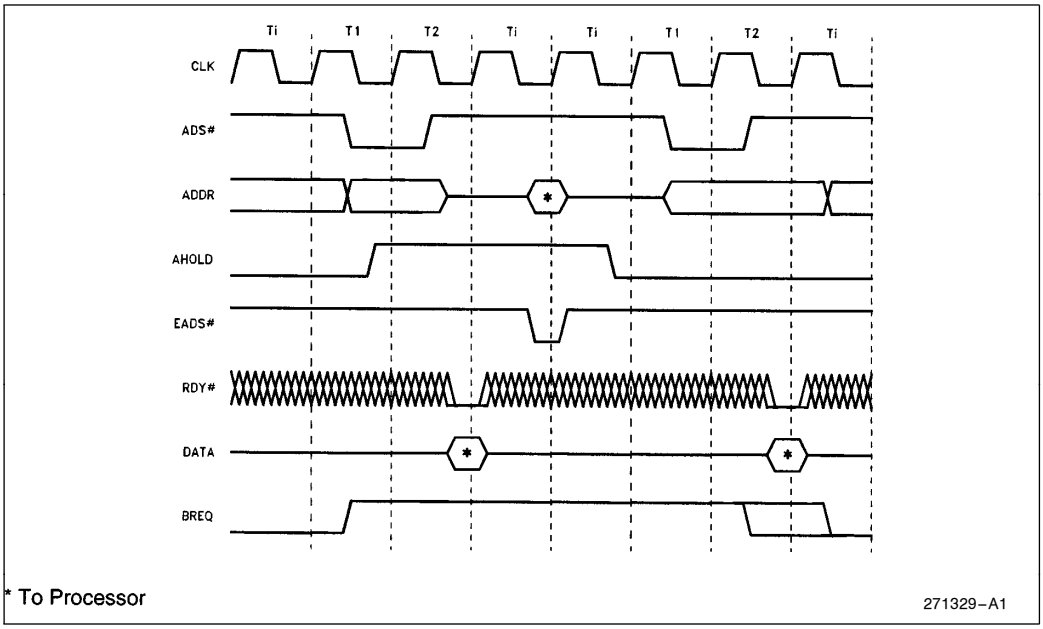


Figure 10-22. Fast Internal Cache Invalidation Cycle

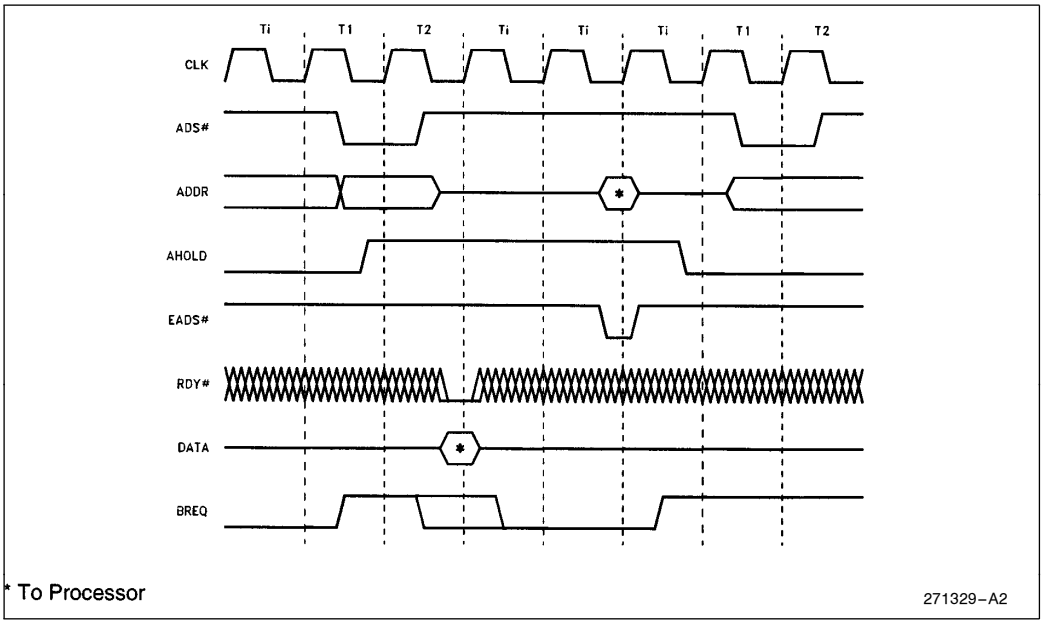


Figure 10-23. Typical Internal Cache Invalidation Cycle

10.2.8.1 Rate of Invalidate Cycles

The Military Intel486 processor can accept one invalidate per clock except in the last clock of a line fill. One invalidate per clock is possible as long as EADS# is negated in ONE or BOTH of the following cases:

1. In the clock RDY# or BRDY# is returned for the last time.
2. In the clock following RDY# or BRDY# being returned for the last time.

This definition allows two system designs. Simple designs can restrict invalidates to one every other clock. The simple design need not track bus activity. Alternatively, systems can request one invalidate per clock provided that the bus is monitored.

10.2.8.2 Running Invalidate Cycles Concurrently with Line Fills

Precautions are necessary to avoid caching stale data in the Military Intel486 processor cache in a system with a second level cache. An example of a system with a second level cache is shown in Figure 10-24.

An external device can be writing to main memory over the system bus while the Military Intel486 processor is retrieving data from the second level cache.

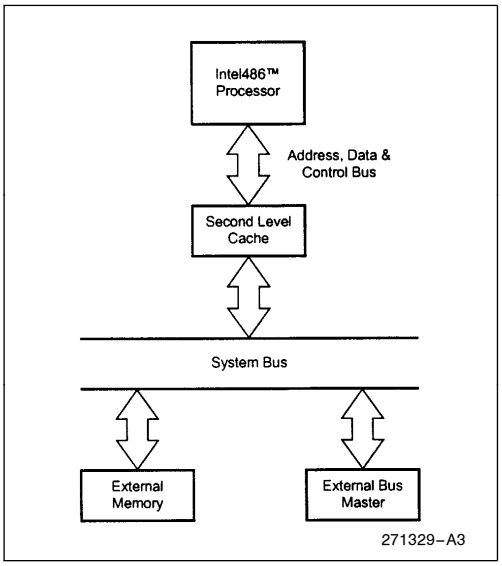


Figure 10-24. System with Second Level Cache

The Military Intel486 processor will need to invalidate a line in its internal cache if the external device is writing to a main memory address also contained in the Military Intel486 processor cache.

A potential problem exists if the external device is writing to an address in external memory, and at the same time the Military Intel486 processor is reading data from the same address in the second level cache. The system must force an invalidation cycle to invalidate the data that the Military Intel486 processor has requested during the line fill.

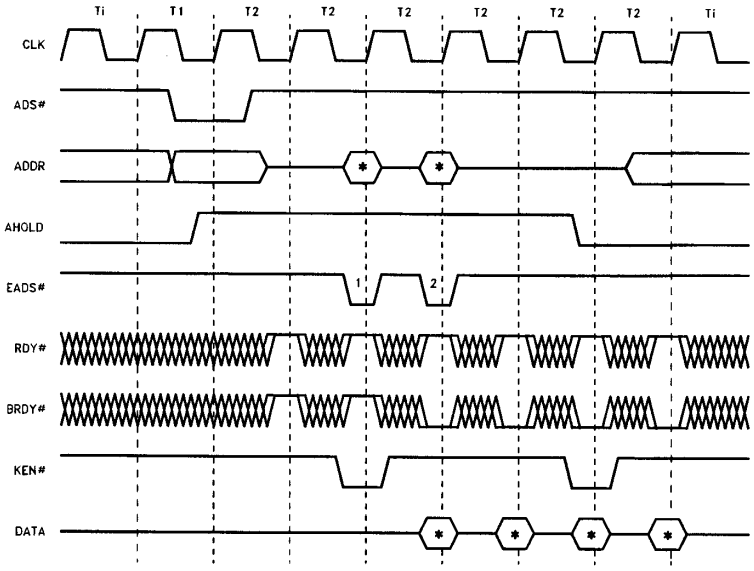
If the system asserts EADS# before the first data in the line fill is returned to the Military Intel486 processor, the system must return data consistent with the new data in the external memory upon resumption of the line fill after the invalidation cycle. This is illustrated by the asserted EADS# signal labeled 1 in Figure 10-25.

If the system asserts EADS# at the same time or after the first data in the line fill is returned (in the same clock that the first RDY# or BRDY# is returned or any subsequent clock in the line fill) the data will be read into the Military Intel486 processor input buffers but it will not be stored in the on-chip cache. This is illustrated by asserted EADS# signal labeled 2 in Figure 10-25. The stale data will be used to satisfy the request that initiated the cache fill cycle.

10.2.9 BUS HOLD

The Military Intel486 processor provides a bus hold, hold acknowledge protocol using the bus hold request (HOLD) and bus hold acknowledge (HLDA) pins. Asserting the HOLD input indicates that another bus master desires control of the Military Intel486 processor bus. The Military Intel486 processor will respond by floating its bus and driving HLDA active when the current bus cycle, or sequence of locked cycles is complete. An example of a HOLD/HLDA transaction is shown in Figure 10-26. Unlike the Intel386 processor, the Military Intel486 processor can respond to HOLD by floating its bus and asserting HLDA while RESET is asserted.

Note that HOLD will be recognized during un-aligned writes (less than or equal to 32-bits) with BLAST# being active for each write. For greater than 32-bit or un-aligned write, HOLD# recognition is prevented by PLOCK# getting asserted. However, HOLD is recognized during non-cacheable, non-burstable code prefetches even though PLOCK# is active.



NOTES:
1. Data returned must be consistent if its address equals the invalidation address in this clock
2. Data returned will not be cached if its address equals the invalidation address in this clock

Figure 10-25. Cache Invalidation Cycle Concurrent with Line Fill

For cacheable and non-bursted or bursted cycles, HOLD is acknowledged during backoff only if HOLD and BOFF# are asserted during an active bus cycle (after ADS# asserted) and before the first RDY# or BRDY# has been returned (see Figure 10-27). The order in which HOLD and BOFF# go active is unimportant (so long as both are active prior to the first RDY#/BRDY# returned by the system). Figure 10-27 shows the case where HOLD is asserted first; HOLD could be asserted simultaneously or after BOFF# and still be acknowledged.

The pins floated during bus hold are: BE0#–BE3#, PCD, PWT, W/R#, D/C#, M/IO#, LOCK#, PLOCK#, ADS#, BLAST#, D0–D31, A2–A31, DP0–DP3.

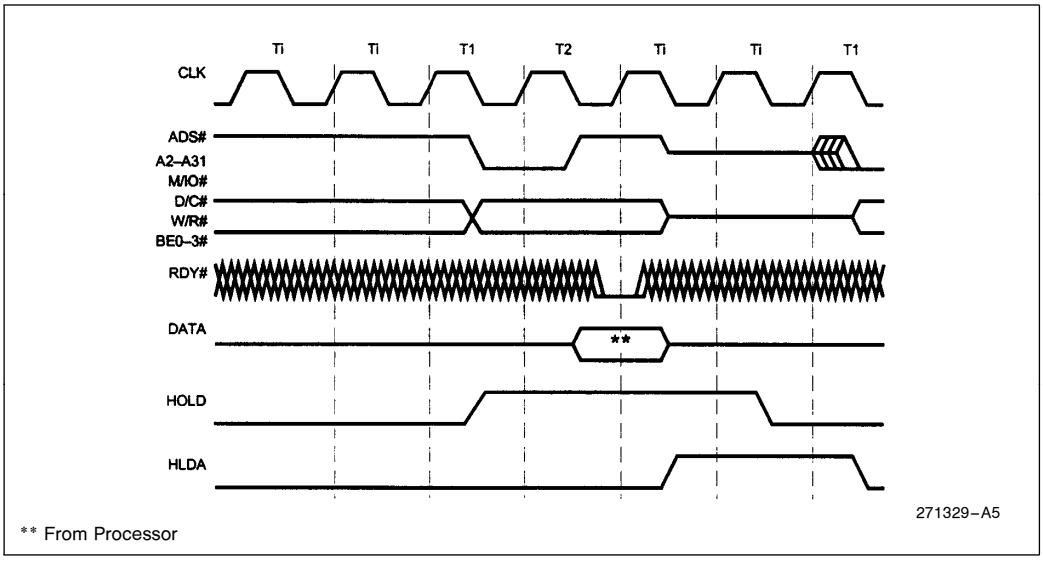


Figure 10-26. HOLD/HLDA Cycles

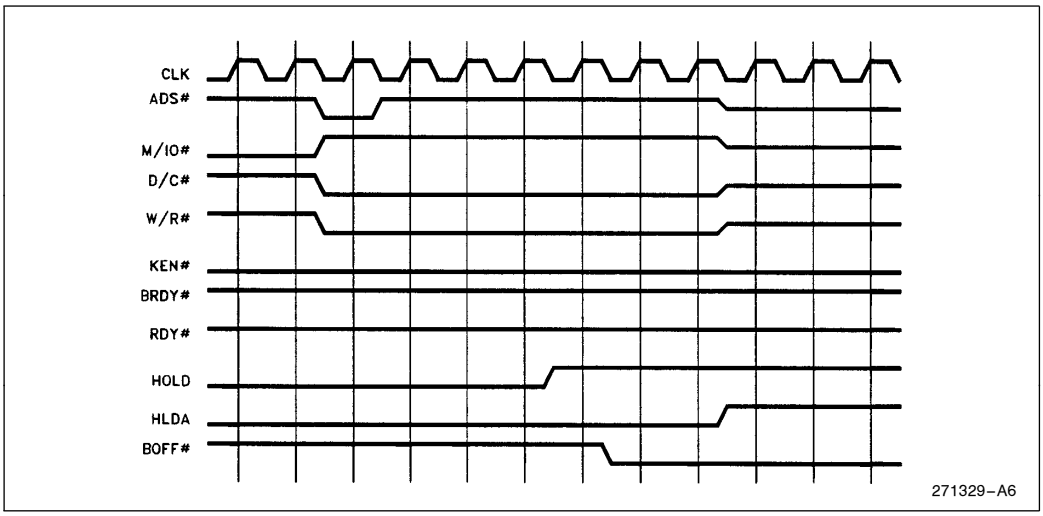


Figure 10-27. HOLD Request Acknowledged during BOFF#

10.2.10 INTERRUPT ACKNOWLEDGE

The Military Intel486 processor generates interrupt acknowledge cycles in response to maskable interrupt requests generated on the interrupt request input (INTR) pin. Interrupt acknowledge cycles have a unique cycle type generated on the cycle type pins.

An example of an interrupt acknowledge transaction is shown in Figure 10-28. Interrupt acknowledge cycles are generated in locked pairs. Data returned during the first cycle is ignored. The interrupt vector is returned during the second cycle on the lower 8 bits of the data bus. The Military Intel486 processor has 256 possible interrupt vectors.

The state of A2 distinguishes the first and second interrupt acknowledge cycles. The byte address driven during the first interrupt acknowledge cycle is 4 (A31–A3 low, A2 high, BE3#–BE1# high, and BE0# low). The address driven during the second interrupt acknowledge cycle is 0 (A31–A2 low, BE3#–BE1# high, BE0# low).

Each of the interrupt acknowledge cycles are terminated when the external system returns RDY# or BRDY#. Wait states can be added by withholding RDY# or BRDY#. The Military Intel486 processor automatically generates four idle clocks between the first and second cycles to allow for 8259A recovery time.

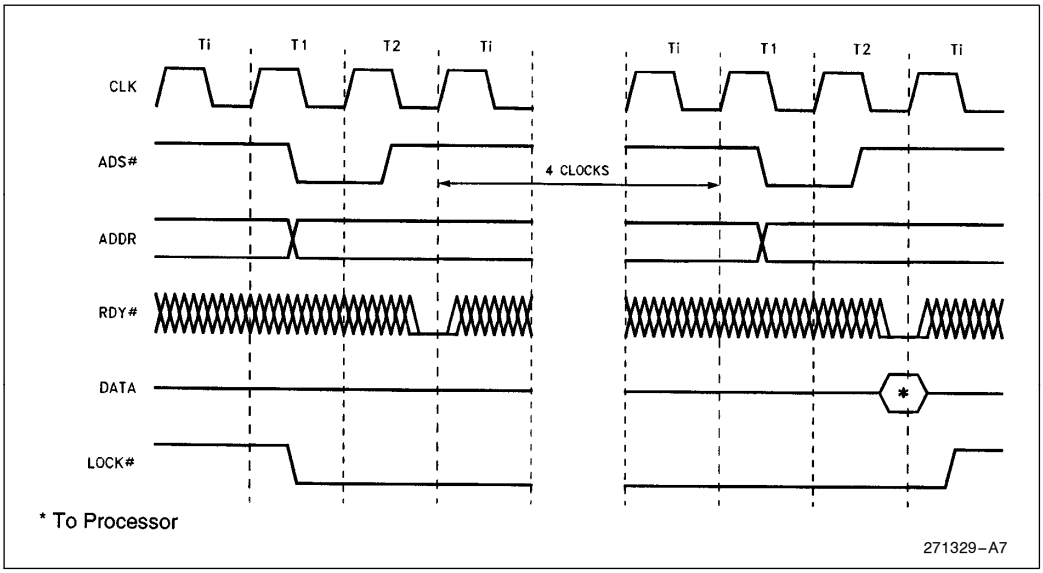


Figure 10-28. Interrupt Acknowledge Cycles

10.2.11 SPECIAL BUS CYCLES

The Military Intel486 processor provides special bus cycles to indicate that certain instructions have been executed, or certain conditions have occurred internally. The special bus cycles in Table 10-9 are defined when the bus cycle definition pins are in the following state: M/IO# = 0, D/C# = 0 and W/R# = 1.

During these cycles the address bus is driven low while the data bus is undefined.

Two of the special cycles indicate halt or shutdown. Another special cycle is generated when the Military Intel486 processor executes an INVD (invalidate data cache) instruction and could be used to flush an external cache. The Write Back cycle is generated when the Military Intel486 processor executes the WBINVD (write-back invalidate data cache) instruction and could be used to synchronize an external write-back cache.

The external hardware must acknowledge these special bus cycles by returning RDY# or BRDY#.

Table 10-9. Special Bus Cycle Encoding

Cycle Name	M/IO#	D/C#	W/R#	BE3#–BE0#	A4–A2
Write-Back	0	0	1	0111	000
Flush	0	0	1	1101	000
Shutdown	0	0	1	1110	000
HALT	0	0	1	1011	000
Stop Grant Ack Cycle	0	0	1	1011	001

NOTE:
1. See section 9.6.1, ''Stop Grant Bus Cycle,'' for details.

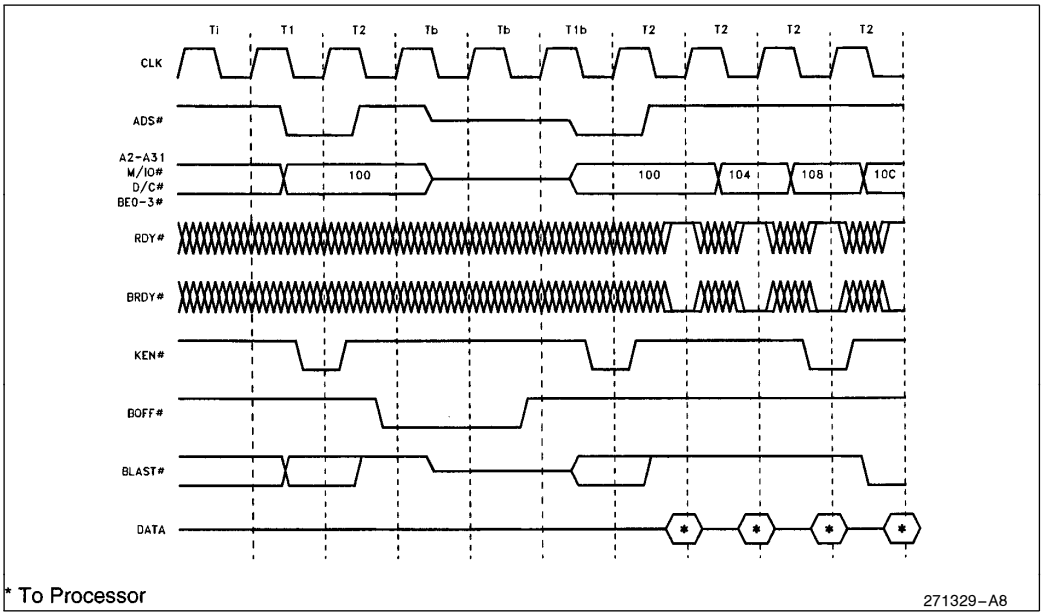


Figure 10-29. Restarted Read Cycle

10.2.11.1 HALT Indication Cycle

The Military Intel486 processor halts as a result of executing a HALT instruction. Signaling its entrance into the HALT state, a HALT indication cycle is performed. The HALT indication cycle is identified by the bus definition signals in special bus cycle state and a byte address of 2. BE0# and BE2# are the only signals distinguishing HALT indication from shutdown indication, which drives an address of 0. During the HALT cycle, undefined data is driven on D0–D31. The HALT indication cycle must be acknowledged by RDY# asserted.

A halted Military Intel486 processor resumes execution when INTR (if interrupts are enabled) or NMI or RESET is asserted.

10.2.11.2 Shutdown Indication Cycle

The Military Intel486 processor shuts down as a result of a protection fault while attempting to process a double fault. Signaling its entrance into the shutdown state, a shutdown indication cycle is performed. The shutdown indication cycle is identified by the bus definition signals in special bus cycle state and a byte address of 0.

10.2.11.3 Stop Grant Indication Cycle

A special Stop Grant bus cycle will be driven to the bus after the processor recognizes the STPCLK# interrupt. The definition of this bus cycle is the same as the HALT cycle definition for the Military Intel486 processor, with the exception that the Stop Grant bus cycle drives the value 0000 0010H on the address pins. The system hardware must acknowledge this cycle by returning RDY# or BRDY#. The processor will not enter the Stop Grant state until either RDY# or BRDY# has been returned. (See Figure 10-31.)

The Stop Grant Bus Cycle is defined as follows:

M/IO# = 0, D/C# = 0, W/R# = 1, Address Bus = 0000 0010H (A_4 = 1), BE3#–BE0# = 1011, Data bus = undefined.

The latency between a STPCLK# request and the Stop Grant bus cycle is dependent on the current instruction, the amount of data in the processor write buffers, and the system memory performance.

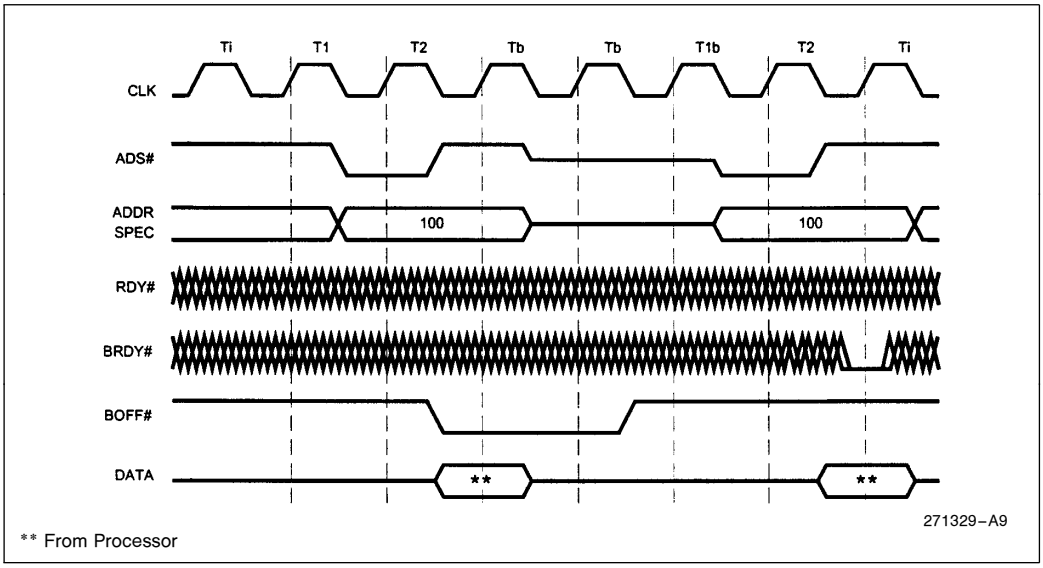


Figure 10-30. Restarted Write Cycle

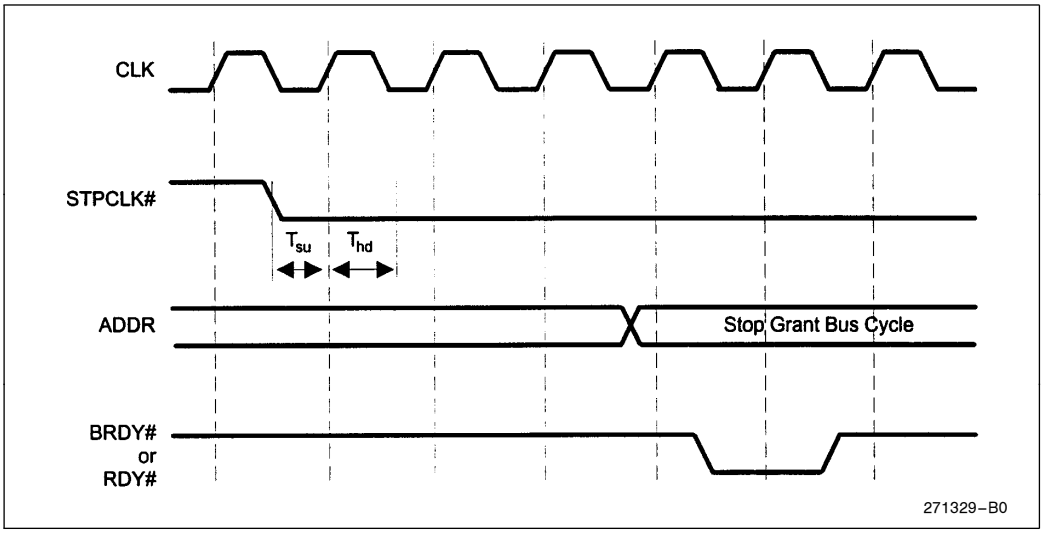


Figure 10-31. Stop Grant Bus Cycle

10.2.12 BUS CYCLE RESTART

In a multi-master system another bus master may require the use of the bus to enable the Military Intel486 processor to complete its current bus request. In this situation the Military Intel486 processor will need to restart its bus cycle after the other bus master has completed its bus transaction.

A bus cycle may be restarted if the external system asserts the backoff (BOFF#) input. The Military Intel486 processor samples the BOFF# pin every clock. The Military Intel486 processor will immediately (in the next clock) float its address, data and status pins when BOFF# is asserted (see Figures 10-29 and 10-34). Any bus cycle in progress when BOFF# is asserted is aborted and any data returned to the processor is ignored. The same pins are floated in response to BOFF# as are floated in response to HOLD. HLDA is not generated in response to BOFF#. BOFF# has higher priority than RDY# or BRDY#. If either RDY# or BRDY# are returned in the same clock as BOFF#, BOFF# takes effect.

The device asserting BOFF# is free to run any cycles it wants while the Military Intel486 processor bus is in its high impedance state. If backoff is requested after the Military Intel486 processor has started a cycle, the new master should wait for memory to return RDY# or BRDY# before assuming control of the bus. Waiting for ready provides a handshake to insure that the memory system is ready to accept a new cycle. If the bus is idle when BOFF# is asserted, the new master can start its cycle two clocks after issuing BOFF#.

The external memory can view BOFF# in the same manner as BLAST#. Asserting BOFF# tells the external memory system that the current cycle is the last cycle in a transfer.

The bus remains in the high impedance state until BOFF# is negated. Upon negation, the Military Intel486 processor restarts its bus cycle by driving out the address and status and asserting ADS#. The bus cycle then continues as usual.

Asserting BOFF# during a burst, BS8# or BS16# cycle will force the Military Intel486 processor to ignore data returned for that cycle only. Data from previous cycles will still be valid. For example, if BOFF# is asserted on the third BRDY# of a burst, the Military Intel486 processor assumes the data returned with the first and second BRDY# is correct and restarts the burst beginning with the third item. The same rule applies to transfers broken into multiple cycle by BS8# or BS16#.

Asserting BOFF# in the same clock as ADS# will cause the Military Intel486 processor to float its bus in the next clock and leave ADS# floating low. Because ADS# is floating low, a peripheral may think that a new bus cycle has begun even-though the cycle was aborted. There are two possible solutions to this problem. The first is to have all devices recognize this condition and ignore ADS# until ready comes back. The second approach is to use a "two clock" backoff: in the first clock AHOLD is asserted, and in the second clock BOFF# is asserted. This guarantees that ADS# will not be floating low. This is only necessary in systems where BOFF# may be asserted in the same clock as ADS#.

10.2.13 BUS STATES

A bus state diagram is shown in Figure 10-32. A description of the signals used in the diagram is given in Table 10-10.

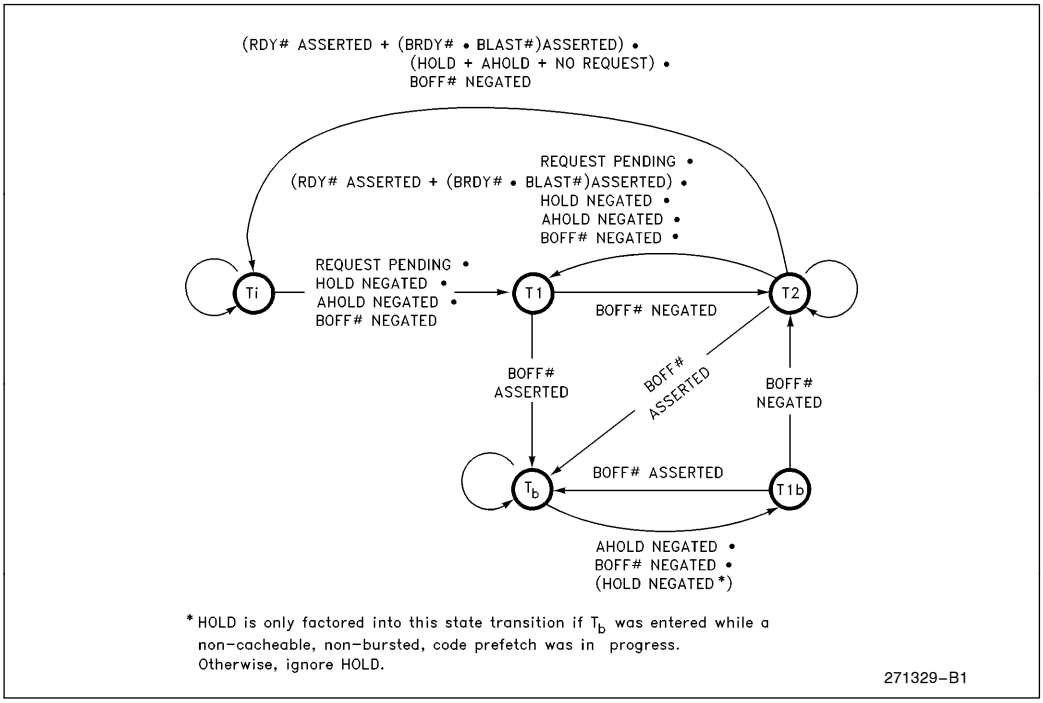


Figure 10-32. Bus State Diagram

Table 10-10. Bus State Description

State	Means
Ti	Bus is idle. Address and status signals may be driven to undefined values, or the bus may be floated to a high impedance state.
T1	First clock cycle of a bus cycle. Valid address and status are driven and ADS# is asserted.
T2	Second and subsequent clock cycles of a bus cycle. Data is driven if the cycle is a write, or data is expected if the cycle is a read. RDY# and BRDY# are sampled.
T1b	First clock cycle of a restarted bus cycle. Valid address and status are driven and ADS# is asserted.
Tb	Second and subsequent clock cycles of an aborted bus cycle.

10.2.14 FLOATING POINT ERROR HANDLING FOR THE MILITARY INTEL486 DX, INTELDX2, AND INTELDX4 PROCESSORS

The Military Intel486 DX, IntelDX2, and IntelDX4 processors provide two options for reporting floating point errors. The simplest method is to raise interrupt 16 whenever an unmasked floating point error occurs. This option may be enabled by setting the NE bit in control register 0 (CR0).

The Military Intel486 DX, IntelDX2, and IntelDX4 processors also provide the option of allowing external hardware to determine how floating point errors are reported. This option is necessary for compatibility with the error reporting scheme used in DOS based systems. The NE bit must be cleared in CR0 to enable user-defined error reporting. User-defined error reporting is the default condition because the NE bit is cleared on reset.

Two pins, floating point error (FERR#) and ignore numeric error (IGNNE#), are provided to direct the actions of hardware if user-defined error reporting is used. The Military Intel486 DX, IntelDX2, and IntelDX4 processors assert the FERR# output to indicate that a floating point error has occurred. FERR# corresponds to the ERROR# pin on the Intel387™ math coprocessor. However, there is a difference in the behavior of the two.

In some cases FERR# is asserted when the next floating point instruction is encountered, and in other cases it is asserted before the next floating point instruction is encountered depending upon the execution state of the instruction causing the exception.

The following class of floating point exceptions drive FERR# at the time the exception occurs (i.e., before encountering the next floating point instruction).

1. The stack fault, invalid operation, and denormal exceptions on all transcendental instructions, integer arithmetic instructions, FSQRT, FSEALE, FPREM(1), FXTRACT, FBLD, and FBSTP.
2. Any exceptions on store instructions (including integer store instructions).

The following class of floating point exceptions drive FERR# only after encountering the next floating point instruction.

1. Exceptions other than on all transcendental instructions, integer arithmetic instructions, FSQRT, FSCALE, FPREM(1), FXTRACT, FBLD, and FBSTP.
2. Any exception on all basic arithmetic, load, compare, and control instructions (i.e., all other instructions).

For both sets of exceptions above, the Intel387 math coprocessor asserts ERROR# when the error occurs and does not wait for the next floating point instruction to be encountered.

IGNNE# is an input to the Military Intel486 DX, IntelDX2, and IntelDX4 processors. When the NE bit in CR0 is cleared, and IGNNE# is asserted, the Military Intel486 DX, IntelDX2, and IntelDX4 processors will ignore a user floating point error and continue executing floating point instructions. When IGNNE# is negated, the IGNNE# is an input to these processors that will freeze on floating point instructions which get errors (except for the control instructions FNCLEX, FNINIT, FNSAVE, FNSTENV, FNSTCW, FNSTSW, FNSTSW AX, FNENI, FNDISI and FNSETPM). IGNNE# may be asynchronous to the Military Intel486 DX, IntelDX2, and IntelDX4 processor clock.

In systems with user-defined error reporting, the FERR# pin is connected to the interrupt controller. When an unmasked floating point error occurs, an interrupt is raised. If IGNNE# is high at the time of this interrupt, the Military Intel486 DX, IntelDX2, and IntelDX4 processors will freeze (disallowing execution of a subsequent floating point instruction) until the interrupt handler is invoked. By driving the IGNNE# pin low (when clearing the interrupt request), the interrupt handler can allow execution of a floating point instruction, within the interrupt handler, before the error condition is cleared (by FNCLEX, FNINIT, FNSAVE or FNSTENV). If execution of a non-control floating point instruction, within the floating point interrupt handler, is not needed, the IGNNE# pin can be tied HIGH.

10.2.15 MILITARY INTEL486 DX, INTELDX2, AND INTELDX4 PROCESSORS FLOATING POINT ERROR HANDLING IN AT-COMPATIBLE SYSTEMS

The Military Intel486 DX, IntelDX2, and IntelDX4 processors provide special features to allow the implementation of an AT-compatible numerics error reporting scheme. These features DO NOT replace the external circuit. Logic is still required that decodes the OUT F0 instruction and latches the FERR# signal. What follows is a description of the use of these Intel Processor features.

The features provided by the Military Intel486 DX, IntelDX2, and IntelDX4 processors are the NE bit in the Machine Status Register, the IGNNE# pin, and the FERR# pin.

The NE bit determines the action taken by the Military Intel486 DX, IntelDX2, and IntelDX4 processors when a numerics error is detected. When set this bit signals that non-DOS compatible error handling will be implemented. In this mode the Military Intel486 DX, IntelDX2, and IntelDX4 processors take a software exception (16) if a numerics error is detected.

If the NE bit is reset, the Military Intel486 DX, IntelDX2, and IntelDX4 processors use the IGNNE# pin to allow an external circuit to control the time at which non-control numerics instructions are allowed to execute. Note that floating point control instructions such as FNINIT and FNSAVE can be executed during a floating point error condition regardless of the state of IGNNE#.

To process a floating point error in the DOS environment the following sequence must take place:

1. The error is detected by the Military Intel486 DX, IntelDX2, and IntelDX4 processor that activates the FERR# pin.
2. FERR# is latched so that it can be cleared by the OUT F0 instruction.
3. The latched FERR# signal activates an interrupt at the interrupt controller. This interrupt is usually handled on IRQ13.
4. The Interrupt Service Routine (ISR) handles the error and then clears the interrupt by executing an OUT instruction to port F0. The address F0 is decoded externally to clear the FERR# latch. The IGNNE# signal is also activated by the decoder output.
5. Usually the ISR then executes an FNINIT instruction or other control instruction before restarting the program. FNINIT clears the FERR# output.

Figure 10-33 illustrates a sample circuit that will perform the function described above. Note that this circuit has not been tested and is included as an example of required error handling logic.

Note that the IGNNE# input allows non-control instructions to be executed prior to the time the FERR# signal is reset by the Military Intel486 DX, IntelDX2, and IntelDX4 processors. This function is implemented to allow exact compatibility with the AT implementation. Most programs reinitialize the floating point unit before continuing after an error is detected. The floating point unit can be reinitialized using one of the following four instructions: FCLEX, FINIT, FSAVE and FSTENV.

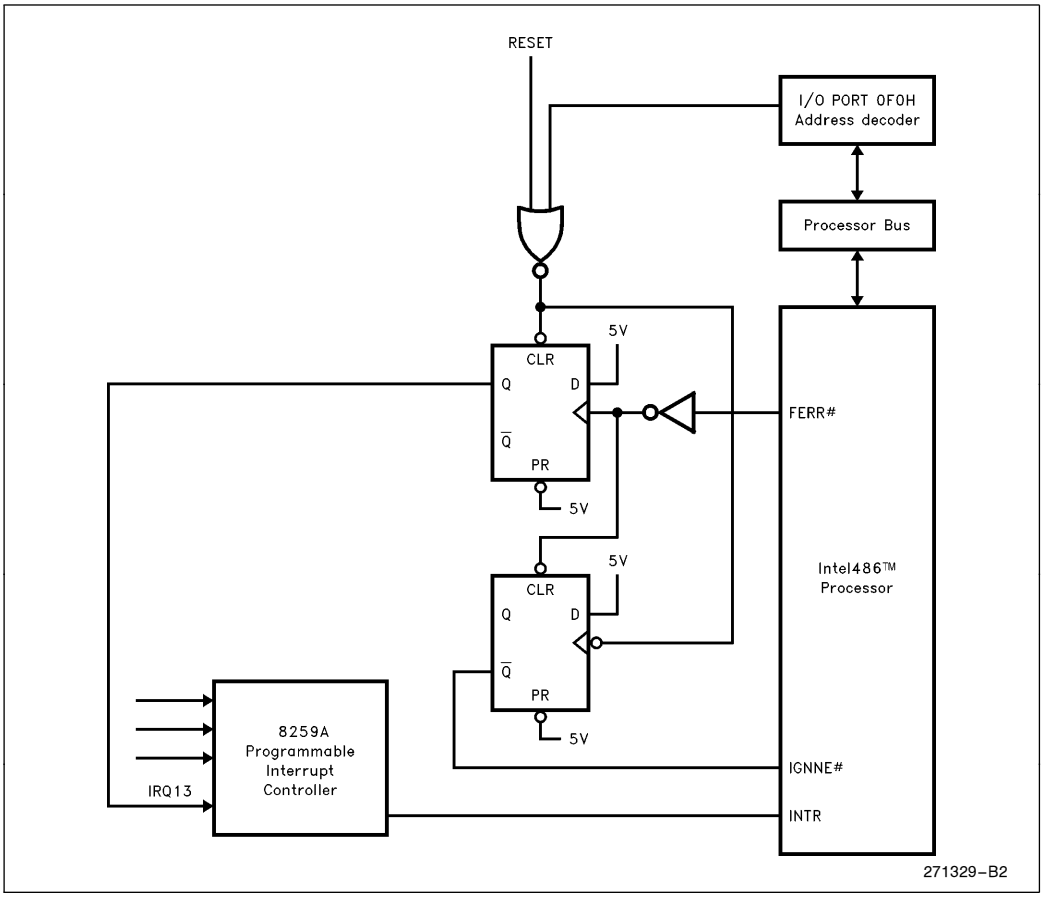


Figure 10-33. DOS-Compatible Numerics Error Circuit

11.0 TESTABILITY

Testing in the Military Intel486 processor can be divided into two categories: Built-in Self Test (BIST) and external testing. The BIST tests the non-random logic, control ROM (CROM), translation lookaside buffer (TLB) and on-chip cache memory. External tests can be run on the TLB and the on-chip cache. The Military Intel486 processor also has a test mode in which all outputs are tri-stated.

11.1 Built-In Self Test (BIST)

The BIST is initiated by holding the AHOLD (address hold) HIGH for 1 CLK after RESET goes from HIGH to LOW, as shown in Figure 9.6. No bus cycles will be run by the Military Intel486 processor until the BIST is concluded. Note that for the Military Intel486 processor, the RESET must be active for 15 clocks with or without BIST enabled for warm resets. SRESET should not be driven active (i.e., high) when entering or during BIST. See Table 11-1 for approximate clocks and maximum completion times for different Military Intel486 processors.

The results of BIST is stored in the EAX register. The Military Intel486 processor has successfully passed the BIST if the contents of the EAX register are zero. If the results in EAX are not zero, then the BIST has detected a flaw in the Military Intel486 processor. The Military Intel486 processor performs reset and begins normal operation at the completion of the BIST.

The non-random logic, control ROM, on-chip cache and translation lookaside buffer (TLB) are tested during the BIST.

The cache portion of the BIST verifies that the cache is functional and that it is possible to read and write to the cache. The BIST manipulates test registers TR3, TR4 and TR5 while testing the cache. These test registers are described in section 11.2, ''On-Chip Cache Testing.''

The cache testing algorithm writes a value to each cache entry, reads the value back, and checks that the correct value was read back. The algorithm may be repeated more than once for each of the 512 cache entries using different constants. The IntelDX4 processor has 1024 cache entries. All other Military Intel486 processors have 512 cache entries.

The TLB portion of the BIST verifies that the TLB is functional and that it is possible to read and write to the TLB. The BIST manipulates test registers TR6 and TR7 while testing the TLB. TR6 and TR7 are described in section 11.3.2, ''TLB Test Registers TR6 and TR7.''

11.2 On-Chip Cache Testing

The on-chip cache testability hooks are designed to be accessible during the BIST and for assembly language testing of the cache.

The Military Intel486 processor contains a cache fill buffer and a cache read buffer. For testability writes, data must be written to the cache fill buffer before it can be written to a location in the cache. Data must be read from a cache location into the cache read buffer before the processor can access the data. The cache fill and cache read buffer are both 128 bits wide.

11.2.1 CACHE TESTING REGISTERS TR3, TR4 AND TR5

Figure 11-1 shows the three cache testing registers: the Cache Data Test Register (TR3), the Cache Status Test Register (TR4) and the Cache Control Test Register (TR5). External access to these registers is provided through MOV reg, TREG and MOV TREG, reg instructions.

Table 11-1. Maximum BIST Completion Time

Processor Type	Core Clock Freq.	Approximate Clocks	Approximate Time for Completions
Military Intel486 DX	33 MHz	1.05 million	32 milliseconds
IntelDX2™	50 MHz	0.6 million	24 milliseconds
IntelDX4™	75 MHz	1.6 million	22 milliseconds

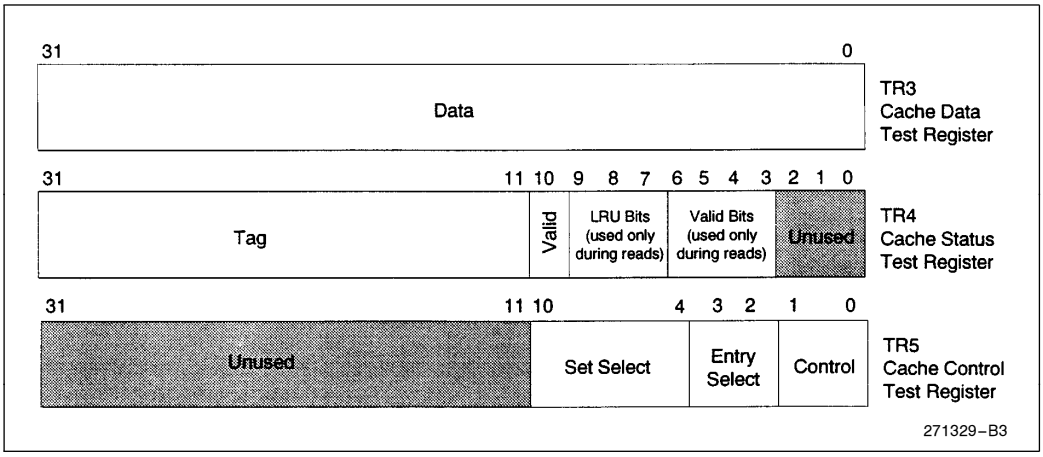


Figure 11-1. Cache Test Registers
(All Military Intel486™ Processors Except the IntelDX4™ Processor)

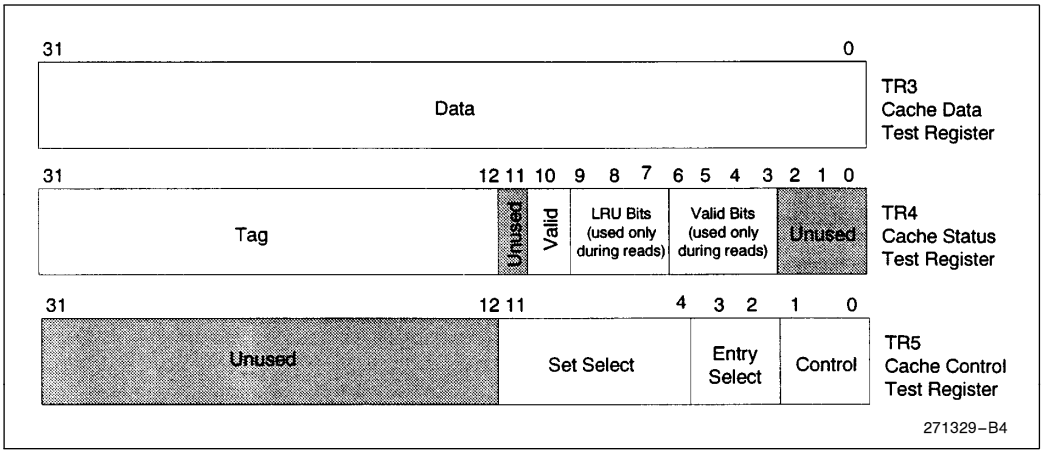


Figure 11-2. IntelDX4™ Processor Cache Test Registers

Cache Data Test Register: TR3

The cache fill buffer and the cache read buffer can only be accessed through TR3. Data to be written to the cache fill buffer must first be written to TR3. Data read from the cache read buffer must be loaded into TR3.

TR3 is 32 bits wide while the cache fill and read buffers are 128 bits wide. 32 bits of data must be written to TR3 four times to fill the cache fill buffer. 32 bits of data must be read from TR3 four times to empty the cache read buffer. The entry select bits in TR5 determine which 32 bits of data TR3 will access in the buffers.

Cache Status Test Register: TR4

TR4 handles tag, LRU and valid bit information during cache tests. TR4 must be loaded with a tag and a valid bit before a write to the cache. After a read from a cache entry, TR4 contains the tag and valid bit from that entry, and the LRU bits and four valid bits from the accessed set. Note that the IntelDX4 processor has one less bit in the TR4 TAG field. (See Figure 11-1.)

Cache Control Test Register: TR5

TR5 specifies which testability operation will be performed and the set and entry within the set which will be accessed. The set select field determines which will be accessed. Note that the IntelDX4 processor has an 8-bit set select field and 256 sets. All other Military Intel486 processors have a 7-bit set select field and 128 sets. (See Figure 11-1.)

The function of the two entry select bits depends on the state of the control bits. When the fill or read buffers are being accessed, the entry select bits point to the 32-bit location in the buffer being accessed. When a cache location is specified, the entry select bits point to one of the four entries in a set. (Refer to Table 11-2.)

Five testability functions can be performed on the cache. The two control bits in TR5 specify the operation to be executed. The five operations are:

1. Write cache fill buffer
2. Perform a cache testability write
3. Perform a cache testability read
4. Read the cache read buffer
5. Perform a cache flush

Table 11-2 shows the encoding of the two control bits in TR5 for the cache testability functions. Table 11-2 also shows the functionality of the entry and set select bits for each control operation.

The cache tests attempt to use as much of the normal operating circuitry as possible. Therefore, when cache tests are being performed, the cache must be disabled (the CD and NW bits in control register 0 (CR0) must be set to 1 to disable the cache. (See section 7.0, ‘‘On-Chip Cache.’’)

11.2.2 CACHE TESTING REGISTERS FOR THE INTELDX4 PROCESSOR

The cache testing registers for the IntelDX4 processor differ slightly from the other Military Intel486 processors. TR3 in the IntelDX4 processor is identical to other Military Intel486 processors. TR4 in the IntelDX4 processor uses bits 31 to 12 for the Tag field, and bit 11 is unused. TR5 uses bits 11 to 4 for the Set Select field. The Test Registers for the IntelDX4 processor are shown in Figure 11-2.

NOTE:

Software written for the Military Intel486 processor for testing the cache using the Test Register will produce failures due to the changes in the TAG bits and Set Select bits for the IntelDX4 processor.

Rewrite the code to take into account the 20 TAG bits and 8 Set Select bits to address the larger cache.

11.2.3 CACHE TESTABILITY WRITE

A testability write to the cache is a two step process. First the cache fill buffer must be loaded with 128 bits of data and TR4 loaded with the tag and valid bit. Next the contents of the fill buffer are written to a cache location.

Loading the fill buffer is accomplished by first writing to the entry select bits in TR5 and setting the control bits in TR5 to 00. The entry select bits identify one of four 32-bit locations in the cache fill buffer to put 32 bits of data. Following the write to TR5, TR3 is written with 32 bits of data which are immediately placed in the cache fill buffer. Writing to TR3 initiates the write to the cache fill buffer. The cache fill buffer is loaded with 128 bits of data by writing to TR5 and TR3 four times using a different entry select location each time.

TR4 must be loaded with the tag and valid bit (bit 10 in TR4) before the contents of the fill buffer are written to a cache location. **The IntelDX4 processor has a 20-bit tag in TR4. All other Military Intel486 processors use a 21-bit tag in TR4.**

The contents of the cache fill buffer are written to a cache location by writing TR5 with a control field of 01 along with the set select and entry select fields. The set select and entry select field indicate the location in the cache to be written. The normal cache LRU update circuitry updates the internal LRU bits for the selected set.

Table 11-2. Cache Control Bit Encoding and Effect of Control Bits on Entry Select and Set Select Functionality

Control Bits		Operation	Entry Select Bits Function	Set Select Bits
Bit 1	Bit 0			
0	0	Enable: Fill Buffer Write Read Buffer Read	Select 32-bit location in fill/read buffer	—
0	1	Perform Cache Write	Select an entry in set	Select a set to write to
1	0	Perform Cache Read	Select an entry in set	Select a set to read from
1	1	Perform Cache Flush	—	—

Note that a cache testability write can only be done when the cache is disabled for replaces (the CD bit is control register 0 is reset to 1). Care must be taken when directly writing to entries in the cache. If the entry is set to overlap an area of memory that is being used in external memory, that cache entry could inadvertently be used instead of the external memory. This is exactly the type of operation that one would desire if the cache were to be used as a high speed RAM. Also, a memory reference (or any external bus cycle) should not occur in between the move to TR4 and the move to TR5, in order to avoid having the value in TR4 change due to the memory reference.

11.2.4 CACHE TESTABILITY READ

A cache testability read is a two step process. First the contents of the cache location are read into the cache read buffer. Next the data is examined by reading it out of the read buffer.

Reading the contents of a cache location into the cache read buffer is initiated by writing TR5 with the control bits set to 10 and the desired set select and two-bit entry select. **The IntelDX4 processor has a seven-bit select field. All other Military Intel486 processors have an eight-bit select field.** In response to the write to TR5, TR4 is loaded with the 21-bit tag field and the single valid bit from the cache entry read. TR4 is also loaded with the three LRU bits and four valid bits corresponding to the cache set that was accessed. The cache read buffer is filled with the 128-bit value which was found in the data array at the specified location.

The contents of the read buffer are examined by performing four reads of TR3. Before reading TR3 the entry select bits in TR5 must loaded to indicate which of the four 32-bit words in the read buffer to transfer into TR3 and the control bits in TR5 must be loaded with 00. The register read of TR3 will initiate the transfer of the 32-bit value from the read buffer to the specified general purpose register.

Note that it is very important that the entire 128-bit quantity from the read buffer and also the information from TR4 be read before any memory references are allowed to occur. If memory operations are allowed to happen, the contents of the read buffer will be corrupted. This is because the testability operations use hardware that is used in normal memory accesses for the Military Intel486 processor whether the cache is enabled or not.

11.2.5 FLUSH CACHE

The control bits in TR5 must be written with 11 to flush the cache. None of the other bits in TR5 have any meaning when 11 is written to the control bits. Flushing the cache will reset the LRU bits and the valid bits to 0, but will not change the cache tag or data arrays.

When the cache is flushed by writing to TR5 the special bus cycle indicating a cache flush to the external system is not run. (See section 10.2.11, ‘‘Special Bus Cycles.’’) For normal operation, the cache should be flushed with the instruction INVD (Invalidate Data Cache) instruction or the WBINVD (Write-back and Invalidate Data Cache) instruction.

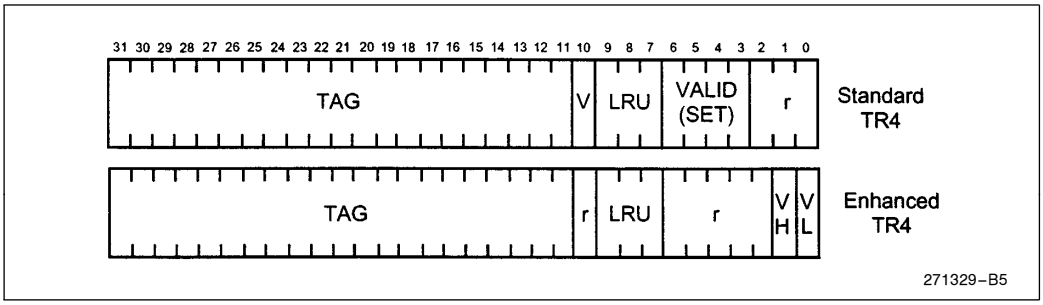


Figure 11-3. TR4 Definition for Standard and Enhanced Bus Modes for the Write-Back Enhanced IntelDX2™Processor

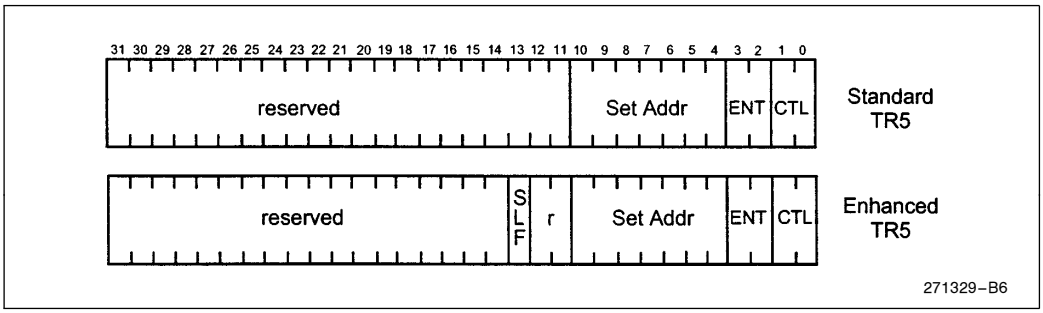


Figure 11-4. TR5 Definition for Standard and Enhanced Bus Modes for the Write-Back Enhanced IntelDX2™ Processor

11.3 Translation Lookaside Buffer (TLB) Testing

The Military Intel486 processor TLB testability hooks are similar to those in the Intel386 processor. The testability hooks have been enhanced to provide added test features and to include new features in the Military Intel486 processor. The TLB testability hooks are designed to be accessible during the BIST and for assembly language testing of the TLB.

11.3.1 TRANSLATION LOOKASIDE BUFFER ORGANIZATION

The Military Intel486 processor TLB is 4-way set associative and has space for 32 entries. The TLB is logically split into three blocks shown in Figure 11-5.

The data block is physically split into four arrays, each with space for eight entries. An entry in the data block is 22 bits wide containing a 20-bit physical address and two bits for the page attributes. The page attributes are the PCD (page cache disable) bit and the PWT (page write-through) bit. Refer to section 7.6, ''Page Cacheability,'' for a discussion of the PCD and PWT bits.

The tag block is also split into four arrays, one for each of the data arrays. A tag entry is 21 bits wide containing a 17-bit linear address and four protection bits. The protection bits are valid (V), user/supervisor (U/S), read/write (R/W) and dirty (D).

The third block contains eight three bit quantities used in the pseudo least recently used (LRU) replacement algorithm. These bits are called the LRU bits. Unlike the on-chip cache, the TLB will replace a valid line even when there is an invalid line in a set.

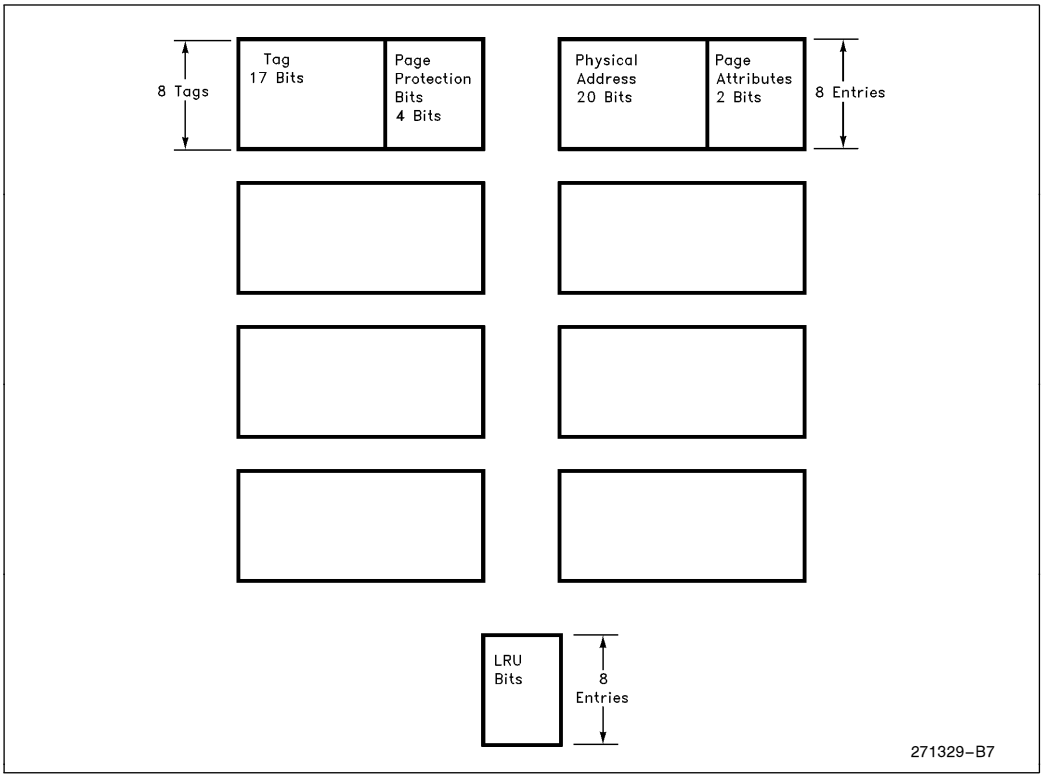


Figure 11-5. TLB Organization

11.3.2 TLB TEST REGISTERS TR6 AND TR7

The two TLB test registers are shown in Figure 11-6. TR6 is the command test register and TR7 is the data test register. External access to these registers is provided through MOV reg,TREG and MOV TREG,reg instructions.

Command Test Register: TR6

TR6 contains the tag information and control information used in a TLB test. Loading TR6 with tag and control information initiates a TLB write or lookup test.

TR6 contains three bit fields, a 20-bit linear address (bits 12–31), seven bits for the TLB tag protection bits (bits 5-11) and one bit (bit 0) to define the type of operation to be performed on the TLB.

The 20-bit linear address forms the tag information used in the TLB access. The lower three bits of the linear address select which of the eight sets are accessed. The upper 17 bits of the linear address form the tag stored in the tag array.

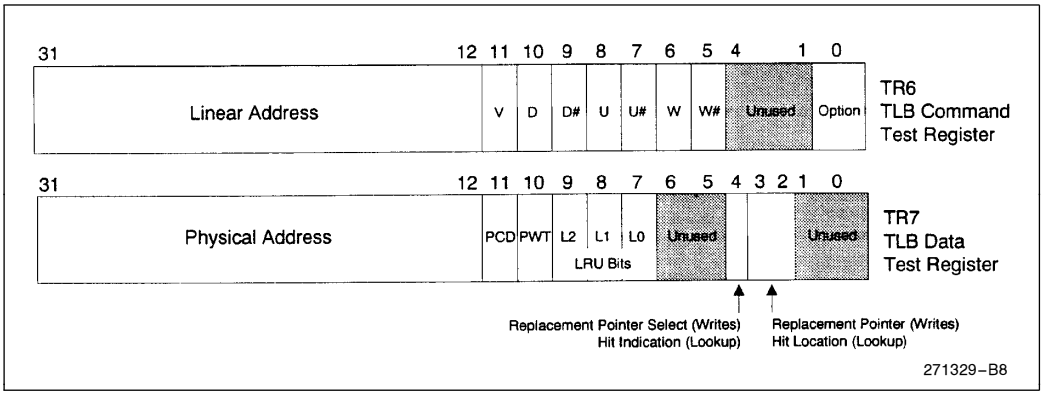


Figure 11-6. TLB Test Registers

The seven TLB tag protection bits are described below.

V: The valid bit for this TLB entry

D,D#: The dirty bit for/from the TLB entry

U,U#: The user/supervisor bit for/from the TLB entry

W,W#: The read/write bit for/from the TLB entry

Two bits are used to represent the D, U/S and R/W bits in the TLB tag to permit the option of a forced miss or hit during a TLB lookup operation. The forced miss or hit will occur regardless of the state of the actual bit in the TLB. The meaning of these pairs of bits is given in Table 11-3.

The operation bit in TR6 determines if the TLB test operation will be a write or a lookup. The function of the operation bit is given in Table 11-4.

Table 11-3. Meaning of a Pair of TR6 Protection Bits

TR6 Protection Bit (B)	TR6 Protection Bit# (B#)	Meaning on TLB Write Operation	Meaning on TLB Lookup Operation
0	0	Undefined	Miss any TLB TAG Bit B
0	1	Write 0 to TLB TAG Bit B	Match TLB TAG Bit B if 0
1	0	Write 1 to TLB TAG Bit B	Match TLB TAG Bit B if 1
1	1	Undefined	Match any TLB TAG Bit B

Table 11-4. TR6 Operation Bit Encoding

TR6 Bit 0	TLB Operation to Be Performed
0	TLB Write
1	TLB Lookup

Data Test Register: TR7

TR7 contains the information stored or read from the data block during a TLB test operation. Before a TLB test write, TR7 contains the physical address and the page attribute bits to be stored in the entry. After a TLB test lookup hit, TR7 contains the physical address, page attributes, LRU bits and entry location from the access.

TR7 contains a 20-bit physical address (bits 12–31), PLD bit (bit 11), PWT bit (bit 10), and three bits for the LRU bits (bits 7–9). The LRU bits in TR7 are only used during a TLB lookup test. The functionality of TR7 bit 4 differs for TLB writes and lookups. The encoding of bit 4 is defined in Table 11-5 and Table 11-6. Finally, TR7 contains two bits (bits 2–3) to specify a TLB replacement pointer or the location of a TLB hit.

Table 11-5. Encoding of Bit 4 of TR7 on Writes

TR7 Bit 4	Replacement Pointer Used on TLB Write
0	Pseudo-LRU Replacement Pointer
1	Data Test Register Bits 3:2

A replacement pointer is used during a TLB write. The pointer indicates which of the four entries in an accessed set is to be written. The replacement pointer can be specified to be the internal LRU bits or bits 2–3 in TR7. The source of the replacement pointer is specified by TR7 bit 4. The encoding of bit 4 during a write is given by Table 11-5.

Note that both testability writes and lookups affect the state of the internal LRU bits regardless of the replacement pointer used. All TLB write operations (testability or normal operation) cause the written entry to become the most recently used. For example, during a testability write with the replacement pointer specified by TR7 bits 2–3, the indicated entry is written and that entry becomes the most recently used as specified by the internal LRU bits.

There are two TLB testing operations: write entries into the TLB, and perform TLB lookups. One major enhancement over TLB testing in the Intel386 processor is that paging need not be disabled while executing testability writes or lookups.

Note that any time one TLB set contains the same linear address in more than one of its entries, looking up that linear address will give unpredictable results. Therefore a single linear address should not be written to one TLB set more than once.

Table 11-6. Encoding of Bit 4 of TR7 on Lookups

TR7 Bit 4	Meaning after TLB Lookup Operation
0	TLB Lookup Resulted in a Miss
1	TLB Lookup Resulted in a Hit

11.3.3 TLB WRITE TEST

To perform a TLB write TR7 must be loaded followed by a TR6 load. The register operations must be performed in this order because the TLB operation is triggered by the write to TR6.

TR7 is loaded with a 20-bit physical address and values for PCD and PWT to be written to the data portion of the TLB. In addition, bit 4 of TR7 must be loaded to indicate whether to use TR7 bits 3-2 or the internal LRU bits as the replacement pointer on the TLB write operation. Note that the LRU bits in TR7 are not used in a write test.

TR6 must be written to initiate the TLB write operation. Bit 0 in TR6 must be reset to zero to indicate a TLB write. The 20-bit linear address and the seven page protection bits must also be written in TR6 to specify the tag portion of the TLB entry. Note that the three least significant bits of the linear address specify which of the eight sets in the data block will be loaded with the physical address data. Thus only 17 of the linear address bits are stored in the tag array.

11.3.4 TLB LOOKUP TEST

To perform a TLB lookup it is only necessary to write the proper tags and control information into TR6. Bit 0 in TR6 must be set to 1 to indicate a TLB lookup. TR6 must be loaded with a 20-bit linear address and the seven protection bits. To force misses and matches of the individual protection bits on TLB lookups, set the seven protection bits as specified in Table 11-3.

A TLB lookup operation is initiated by the write to TR6. TR7 will indicate the result of the lookup operation following the write to TR6. The hit/miss indication can be found in TR7 bit 4 (see Table 11-6).

TR7 will contain the following information if bit 4 indicated that the lookup test resulted in a hit. Bits 2–3 will indicate in which set the match occurred. The 22 most significant bits in TR7 will contain the physical address and page attributes contained in the entry. Bits 9–7 will contain the LRU bits associated with the accessed set. The state of the LRU bits is previous to their being updated for the current lookup.

If bit 4 in TR7 indicated that the lookup test resulted in a miss the remaining bits in TR7 are undefined.

Again it should be noted that a TLB testability lookup operation affects the state of the LRU bits. The LRU bits will be updated if a hit occurred. The entry which was hit will become the most recently used.

11.4 Tri-State Output Test Mode

The Military Intel486 processor provides the ability to float all its outputs and bidirectional pins, except for the VOLDET pin in the IntelDX4 processor. This includes all pins floated during bus hold as well as pins which are never floated in normal operation of the chip (HLDA, BREQ, FERR# and PCHK#). When the Military Intel486 processor is in the tri-state output test mode external testing can be used to test board connections.

The tri-state test mode is invoked if FLUSH# is sampled active at the falling edge of RESET. FLUSH# is an asynchronous signal. When driven, FLUSH# should be asserted for 2 clocks before and 2 clocks after RESET is de-asserted. If FLUSH# is driven synchronously, the tri-state output test mode is initiated by driving FLUSH# so that it is sampled active in the clock prior to RESET going low and ensuring that specified setup and hold times are met. The outputs are guaranteed to tri-state no later than 10 clocks after RESET goes low (see Figure 9.6). The Military Intel486 processor remains in the tri-state test mode until the next RESET.

11.5 Military Intel486 Processor Boundary Scan (JTAG)

The Military Intel486 processor provides additional testability features compatible with the IEEE Standard Test Access Port and Boundary Scan Architecture (IEEE Std. 1149.1). (Note that the Military Intel486 SX processor in PGA package does *not* have JTAG capability.) The test logic provided allows for testing to insure that components function correctly, that interconnections between various components are correct, and that various components interact correctly on the printed circuit board.

The boundary scan test logic consists of a boundary scan register and support logic that are accessed through a test access port (TAP). The TAP provides a simple serial interface that makes it possible to test all signal traces with only a few probes.

The TAP can be controlled via a bus master. The bus master can be either automatic test equipment or a component (PLD) that interfaces to the four-pin test bus.

11.5.1 BOUNDARY SCAN ARCHITECTURE

The boundary scan test logic contains the following elements:

- Test access port (TAP), consisting of input pins TMS, TCK, and TDI; and output pin TDO.
- TAP controller, which interprets the inputs on the test mode select (TMS) line and performs the corresponding operation. The operations performed by the TAP include controlling the instruction and data registers within the component.
- Instruction register (IR), which accepts instruction codes shifted into the test logic on the test data input (TDI) pin. The instruction codes are used to select the specific test operation to be performed or the test data register to be accessed.
- Test data registers: The Military Intel486 processor contains three test data registers: Bypass register (BPR), Device Identification register (DID), and Boundary Scan register (BSR).

The instruction and test data registers are separate shift-register paths connected in parallel and have a common serial data input and a common serial data output connected to the TAP signals, TDI and TDO, respectively.

11.5.2 DATA REGISTERS

The Military Intel486 processor contains the two required test data registers; bypass register and

boundary scan register. In addition, they also have a device identification register.

Each test data register is serially connected to TDI and TDO, with TDI connected to the most significant bit and TDO connected to the least significant bit of the test data register.

Data is shifted one stage (bit position within the register) on each rising edge of the test clock (TCK). In addition the Military Intel486 processor contains a runbist register to support the RUNBIST boundary scan instruction.

11.5.2.1 Bypass Register

The Bypass Register is a one-bit shift register that provides the minimal length path between TDI and TDO. This path can be selected when no test operation is being performed by the component to allow rapid movement of test data to and from other components on the board. While the bypass register is selected data is transferred from TDI to TDO without inversion.

11.5.2.2 Boundary Scan Register

The Boundary Scan Register is a single shift register path containing the boundary scan cells that are connected to all input and output pins of the Military Intel486 processor. Figure 11-7 shows the logical structure of the boundary scan register. While output cells determine the value of the signal driven on the corresponding pin, input cells only capture data; they do not affect the normal operation of the device. Data is transferred without inversion from TDI to TDO through the boundary scan register during scanning. The boundary scan register can be operated by the EXTEST and SAMPLE instructions. The boundary scan register order is described in section 11.5.5 ''Boundary Scan Register Bits and Bit Orders.''

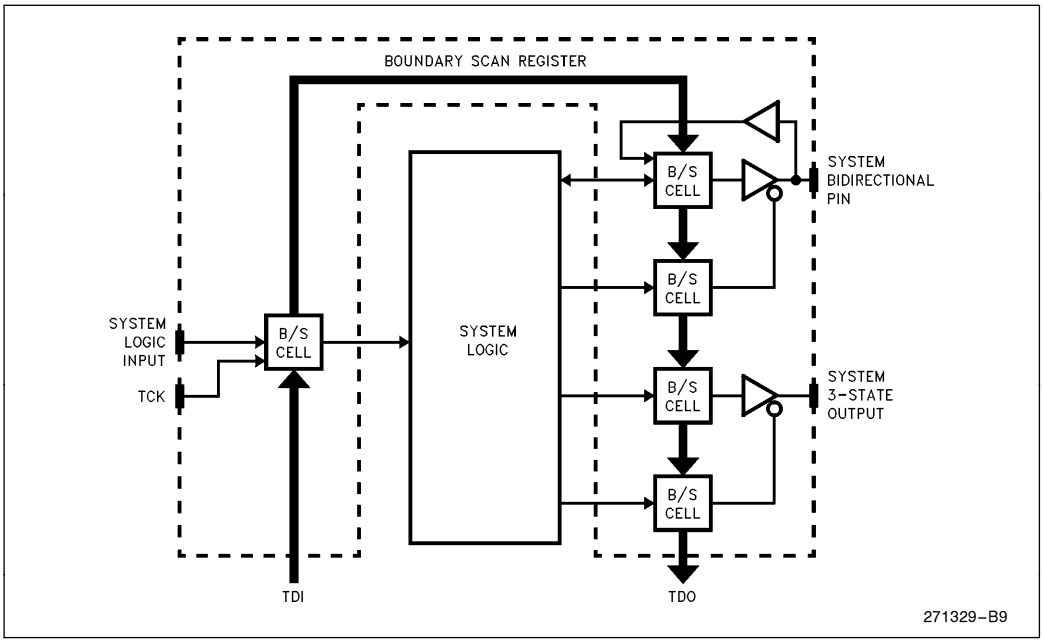


Figure 11-7. Logical Structure of Boundary Scan Register

11.5.2.3 Device Identification Register

The Device Identification Register contains the manufacturer's identification code, part number code, and version code. Table 11-7 lists the codes corresponding to the Military Intel486 processor.

11.5.2.4 Runbist Register

The Runbist Register is a one bit register used to report the results of the Military Intel486 processor BIST when it is initiated by the RUNBIST instruction. This register is loaded with a ''1'' prior to invoking the BIST and is loaded with ''0'' upon successful completion.

11.5.3 INSTRUCTION REGISTER

The Instruction Register (IR) allows instructions to be serially shifted into the device. The instruction selects the particular test to be performed, the test data register to be accessed, or both. The instruction register is four (4) bits wide. The most significant bit is connected to TDI and the least significant bit is connected to TDO. There are no parity bits associated with the Instruction register. Upon entering the Capture-IR TAP controller state, the Instruction register is loaded with the default instruction ''0001,'' SAMPLE/PRELOAD. Instructions are shifted into the instruction register on the rising edge of TCK while the TAP controller is in the SHIFT-IR state.

11.5.3.1 Boundary Scan Instruction Set

The Military Intel486 processor supports all three mandatory boundary scan instructions (BYPASS, SAMPLE/PRELOAD, and EXTEST) along with two optional instructions (IDCODE and RUNBIST). Table 11-8 lists the Military Intel486 processor boundary scan instruction codes. The instructions listed as PRIVATE cause TDO to become enabled in the Shift-DR state and cause '0' to be shifted out of TDO on the rising edge of TCK. Execution of the PRIVATE instructions will not cause hazardous operation of the Military Intel486 processor.

EXTEST The instruction code is ''0000.'' The EXTEST instruction allows testing of circuitry external to the component package, typically board interconnects. It does so by driving the values loaded into the Military Intel486 processor's boundary scan register out on the output pins corresponding to each boundary scan cell and capturing the values on Military Intel486 processor input pins to be loaded into their corresponding boundary scan register locations. I/O pins are selected as input or output, depending on the value loaded into their control setting locations in the boundary scan register. Values shifted into input latches in the boundary scan register are never used by the internal logic of the Military Intel486 processor.

Table 11-7. Boundary Scan Component Identification Codes

Processor Type	Version	V_{CC} 1 = 3.3V 0 = 5V	Intel Architecture Type	Family	Model	MFG ID Intel = 009H	1st Bit	Boundary Scan ID (Hex)
Military Intel486 DX processor (5V)	xxxx*	0	000001	0100	00001	00000001001	1	x0281013H
IntelDX2 processor (5V)	xxxx*	0	000001	0100	00101	00000001001	1	x0285013H
IntelDX4™ processor (3.3V)	xxxx*	1	000001	0100	01000	00000001001	1	x8288013H

NOTE:
*Contact Intel for details

Table 11-8. Boundary Scan Instruction Codes

Instruction Code	Instruction Name
0000	EXTEST
0001	SAMPLE
0010	IDCODE
0011	PRIVATE
0100	PRIVATE
0101	PRIVATE
0110	PRIVATE
0111	PRIVATE
1000	RUNBIST
1001	PRIVATE
1010	PRIVATE
1011	PRIVATE
1100	PRIVATE
1101	PRIVATE
1110	PRIVATE
1111	BYPASS

NOTE:
After using the EXTEST instruction, the Military Intel486 processor must be reset before normal (non-boundary scan) use.

SAMPLE/PRELOAD The instruction code is ''0001.'' The SAMPLE/PRELOAD has two functions that it performs. When the TAP controller is in the Capture-DR state, the SAMPLE/PRELOAD instruction allows a ''snap-shot'' of the normal operation of the component without interfering with that normal operation. The instruction causes boundary scan register cells associated with outputs to sample the value being driven by the Military Intel486 processor. It causes the cells associated with inputs to sample the value being driven into the Military Intel486 processor. On both outputs and inputs the sampling occurs on the rising edge of TCK. When the TAP controller is in the Update-DR state, the SAMPLE/PRELOAD instruction preloads data to the device pins to be driven to the board by executing the EXTEST instruction. Data is preloaded to the pins from the boundary scan register on the falling edge of TCK.

IDODE The instruction code is ''0010.'' The IDCODE instruction selects the device identification register to be connected to TDI and TDO, allowing the device identification code to be shifted out of the device on TDO. Note that the device identification register is not altered by data being shifted in on TDI.

BYPASS The instruction code is ''1111.'' The BYPASS instruction selects the bypass register to be connected to TDI and TDO, effectively bypassing the test logic on the Military Intel486 processor by reducing the shift length of the device to one bit. Note that an open circuit fault in the board level test data path will cause the bypass register to be selected following an instruction scan cycle due to the pull-up resistor on the TDI input. This has been done to prevent any unwanted interference with the proper operation of the system logic.

RUNBIST The instruction code is ''1000.'' The RUNBIST instruction selects the one (1) bit runbist register, loads a value of ''1'' into the runbist register, and connects it to TDO. It also initiates the built-in self test (BIST) feature of the Military Intel486 processor, which is able to detect approximately 60% of the stuck-at faults on the Military Intel486 processor. The Military Intel486 processor ac/dc specifications for V_{CC} and CLK must be met and RESET must have been asserted at least once prior to executing the RUNBIST boundary scan instruction. After loading the RUNBIST instruction code in the instruction register, the TAP controller must be placed in the Run-Test/Idle state. BIST begins on the first rising edge of TCK after entering the Run-Test/Idle state. The TAP controller must remain in the Run-Test/Idle state until BIST is completed. It requires 1.2 million clock (CLK) cycles to complete BIST and report the result to the runbist register. After completing the 1.2 million clock (CLK) cycles, the value in the runbist register should be shifted out on

TDO during the Shift-DR state. A value of ‘‘0’’ being shifted out on TDO indicates BIST successfully completed. A value of ‘‘1’’ indicates a failure occurred. After executing the RUNBIST instruction, the Military Intel486 processor must be reset prior to normal operation.

11.5.4 TEST ACCESS PORT (TAP) CONTROLLER

The TAP controller is a synchronous, finite state machine. It controls the sequence of operations of the test logic. The TAP controller changes state only in response to the following events:

1. a rising edge of TCK
2. power-up.

The value of the test mode state (TMS) input signal at a rising edge of TCK controls the sequence of the state changes. The state diagram for the TAP controller is shown in Figure 11-8. Test designers must consider the operation of the state machine in order to design the correct sequence of values to drive on TMS.

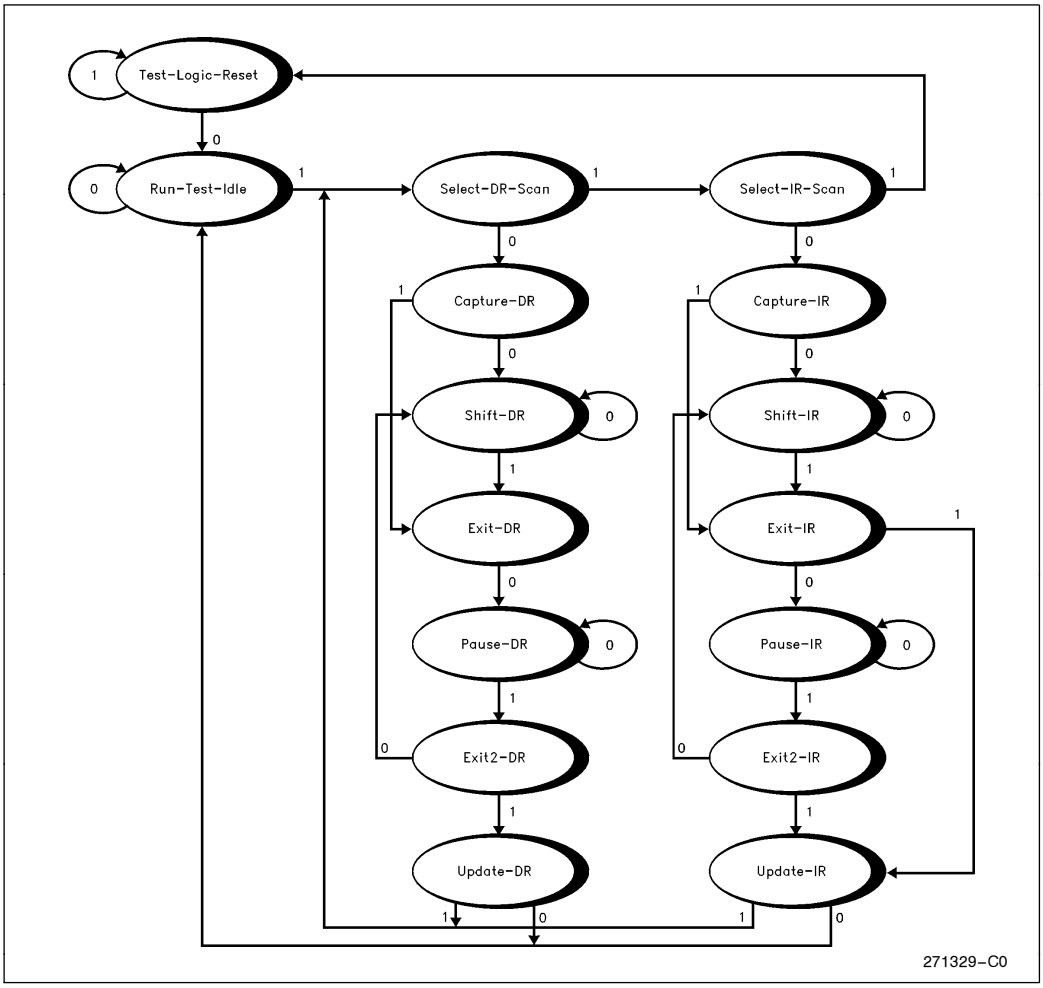


Figure 11-8. TAP Controller State Diagram

11.5.4.1 Test-Logic-Reset State

In this state, the test logic is disabled so that normal operation of the device can continue unhindered. This is achieved by initializing the instruction register such that the IDCODE instruction is loaded. No matter what the original state of the controller, the controller enters Test-Logic-Reset state when the TMS input is held high (1) for at least five rising edges of TCK. The controller remains in this state while TMS is high. The TAP controller is also forced to enter this state at power-up.

11.5.4.2 Run-Test/Idle State

A controller state between scan operations. Once in this state, the controller remains in this state as long as TMS is held low. In devices supporting the RUNBIST instruction, the BIST is performed during this state and the result is reported in the runbist register. For instruction not causing functions to execute during this state, no activity occurs in the test logic. The instruction register and all test data registers retain their previous state. When TMS is high and a rising edge is applied to TCK, the controller moves to the Select-DR state.

11.5.4.3 Select-DR-Scan State

This is a temporary controller state. The test data register selected by the current instruction retains its previous state. If TMS is held low and a rising edge is applied to TCK when in this state, the controller moves into the Capture-DR state, and a scan sequence for the selected test data register is initiated. If TMS is held high and a rising edge is applied to TCK, the controller moves to the Select-IR-Scan state.

The instruction does not change in this state.

11.5.4.4 Capture-DR State

In this state, the boundary scan register captures input pin data if the current instruction is EXTEST or SAMPLE/PRELOAD. The other test data registers, which do not have parallel input, are not changed.

The instruction does not change in this state.

When the TAP controller is in this state and a rising edge is applied to TCK, the controller enters the Exit1-DR state if TMS is high or the Shift-DR state if TMS is low.

11.5.4.5 Shift-DR State

In this controller state, the test data register connected between TDI and TDO as a result of the current instruction shifts data one stage toward its serial output on each rising edge of TCK.

The instruction does not change in this state.

When the TAP controller is in this state and a rising edge is applied to TCK, the controller enters the Exit1-DR state if TMS is high or remains in the Shift-DR state if TMS is low.

11.5.4.6 Exit1-DR State

This is a temporary state. While in this state, if TMS is held high, a rising edge applied to TCK causes the controller to enter the Update-DR state, which terminates the scanning process. If TMS is held low and a rising edge is applied to TCK, the controller enters the Pause-DR state.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

11.5.4.7 Pause-DR State

The pause state allows the test controller to temporarily halt the shifting of data through the test data register in the serial path between TDI and TDO. An example of using this state could be to allow a tester to reload its pin memory from disk during application of a long test sequence.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

The controller remains in this state as long as TMS is low. When TMS goes high and a rising edge is applied to TCK, the controller moves to the Exit2-DR state.

11.5.4.8 Exit2-DR State

This is a temporary state. While in this state, if TMS is held high, a rising edge applied to TCK causes the controller to enter the Update-DR state, which terminates the scanning process. If TMS is held low and a rising edge is applied to TCK, the controller enters the Shift-DR state.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

11.5.4.9 Update-DR State

The boundary scan register is provided with a latched parallel output to prevent changes at the parallel output while data is shifted in response to the EXTEST and SAMPLE/PRELOAD instructions. When the TAP controller is in this state and the boundary scan register is selected, data is latched onto the parallel output of this register from the shift-register path on the falling edge of TCK. The data held at the latched parallel output does not change other than in this state.

All test data registers selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

11.5.4.10 Select-IR-Scan State

This is a temporary controller state. The test data register selected by the current instruction retains its previous value. If TMS is held low and a rising edge is applied to TCK when in this state, the controller moves into the Capture-IR state, and a scan sequence for the instruction register is initiated. If TMS is held high and a rising edge is applied to TCK, the controller moves to the Test-Logic-Reset state.

The instruction does not change in this state.

11.5.4.11 Capture-IR State

In this controller state the shift register contained in the instruction register loads the fixed value ‘‘0001’’ on the rising edge of TCK.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state. When the controller is in this state and a rising edge is applied to TCK, the controller enters the Exit1-IR state if TMS is held high, or the Shift-IR state if TMS is held low.

11.5.4.12 Shift-IR State

In this state the shift register contained in the instruction register is connected between TDI and TDO and shifts data one stage towards its serial output on each rising edge of TCK.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

When the controller is in this state and a rising edge is applied to TCK, the controller enters the Exit1-IR state if TMS is held high, or remains in the Shift-IR state if TMS is held low.

11.5.4.13 Exit1-IR State

This is a temporary state. While in this state, if TMS is held high, a rising edge applied to TCK causes the controller to enter the Update-IR state, which terminates the scanning process. If TMS is held low and a rising edge is applied to TCK, the controller enters the Pause-IR state.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

11.5.4.14 Pause-IR State

The pause state allows the test controller to temporarily halt the shifting of data through the instruction register.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

The controller remains in this state as long as TMS is low. When TMS goes high and a rising edge is applied to TCK, the controller moves to the Exit2-IR state.

11.5.4.15 Exit2-IR State

This is a temporary state. While in this state, if TMS is held high, a rising edge applied to TCK causes the controller to enter the Update-IR state, which terminates the scanning process. If TMS is held low and a rising edge is applied to TCK, the controller enters the Shift-IR state.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

11.5.4.16 Update-IR State

The instruction shifted into the instruction register is latched onto the parallel output from the shift-register path on the falling edge of TCK. Once the new instruction has been latched, it becomes the current instruction.

Test data registers selected by the new current instruction retain the previous value.

11.5.5 BOUNDARY SCAN REGISTER BITS AND BIT ORDERS

The boundary scan register contains a cell for each pin, as well as cells for control of I/O and tri-state pins.

Military Intel486 DX and IntelDX2 Processor Boundary Scan Register Bits

The following is the bit order of the Military Intel486 DX and IntelDX2 processor boundary scan register (from left to right and top to bottom. See notes below):

TDO ← A2, A3, A4, A5, UP#, A6, A7, A8, A9, A10, A11, A12, A13, A14, A15, A16, A17, A18, A19, A20, A21, A22, A23, A24, A25, A26, A27, A28, A29, A30, A31, DP0, D0, D1, D2, D3, D4, D5, D6, D7, DP1, D8, D9, D10, D11, D12, D13, D14, D15, DP2, D16, D17, D18, D19, D20, D21, D22, D23, DP3, D24, D25, D26, D27, D28, D29, D30, D31, STPCLK#, IGNNE#, FERR#, SMI#, SMIACT#, SRESET, NMI, INTR, FLUSH#, RESET, A20M#, EADS#, PCD, PWT, D/C#, M/IO#, BE3#, BE2#, BE1#, BE0#, BREQ, W/R#, HLDA, CLK, Reserved, AHOLD, HOLD, KEN#, RDY#, BS8#, BS16#, BOFF#, BRDY#, PCHK#, LOCK#, PLOCK#, BLAST#, ADS#, MISCCTL, BUSCTL, ABUSCTL, WRTL ← TDI

IntelDX4 Processor Boundary Scan Register Bits

The following is the bit order of the IntelDX4 processor boundary scan register (from left to right and top to bottom. See notes below):

TDO ← A2, A3, A4, A5, UP#, A6, A7, A8, A9, A10, A11, A12, A13, A14, A15, A16, A17, A18, A19, A20, A21, A22, A23, A24, A25, A26, A27, A28, A29, A30, A31, DP0, D0, D1, D2, D3, D4, D5, D6, D7, DP1, D8, D9, D10, D11, D12, D13, D14, D15, DP2, D16, D17, D18, D19, D20, D21, D22, D23, DP3, D24, D25, D26, D27, D28, D29, D30, D31, STPCLK#, IGNNE#, FERR#, SMI#, SMIACT#, SRESET, NMI, INTR, FLUSH#, RESET, A20M#, EADS#, PCD, PWT, D/C#, M/IO#, BE3#, BE2#, BE1#, BE0#, BREQ, W/R#, HLDA, CLK, AHOLD, HOLD, KEN#, RDY#, CLKMUL, BS8#, BS16#, BOFF#, BRDY#, PCHK#, LOCK#, PLOCK#, BLAST#, ADS#, MISCCTL, BUSCTL, ABUSCTL, WRTL ← TDI

NOTES:

''Reserved'' corresponds to no connect ''NC'' or ''INC'' signals on the Military Intel486 processor.

All the *CTL cells are control cells that are used to select the direction of bidirectional pins or tri-state output pins. If '1' is loaded into the control cell (*CTL), the associated pin(s) are tri-stated or selected as input. The following lists the control cells and their corresponding pins.

1. WRCTL controls the D31–D0 and DP3–DP0 pins.
2. ABUSCTL controls the A31–A2 pins.
3. BUSCTL controls the ADS#, BLAST#, PLOCK#, LOCK#, WR#, BE0#, BE1#, BE2#, BE3#, MIO#, DC#, PWT, and PCD pins.
4. MISCCTL controls the PCHK#, HLDA, and BREQ pins.

11.5.6 TAP CONTROLLER INITIALIZATION

The TAP controller is automatically initialized when a device is powered up. In addition, the TAP controller can be initialized by applying a high signal level on the TMS input for five TCK periods.

11.5.7 BOUNDARY SCAN DESCRIPTION LANGUAGE (BSDL) FILES

See Appendix C for an example of a BSDL file for Military Intel486 processors.

12.0 DEBUGGING SUPPORT

The Military Intel486 processor provides several features that simplify the debugging process. The three categories of on-chip debugging aids are:

1. Code execution breakpoint opcode (0CCH),
2. Single-step capability provided by the TF bit in the flag register, and
3. Code and data breakpoint capability provided by the Debug Registers DR0–3, DR6, and DR7.

12.1 Breakpoint Instruction

A single-byte-opcode breakpoint instruction is available for use by software debuggers. The breakpoint opcode is 0CCH, and generates an exception 3 trap when executed. In typical use, a debugger program can ''plant'' the breakpoint instruction at all desired code execution breakpoints. The single-byte breakpoint opcode is an alias for the two-byte general software interrupt instruction, INT n, where n = 3. The only difference between INT 3 (0CCh) and INT n is that INT 3 is never IOPL-sensitive, while INT n is IOPL-sensitive in Protected Mode and Virtual 8086 Mode.

12.2 Single-Step Trap

If the single-step flag (TF, bit 8) in the EFLAG register is found to be set at the end of an instruction, a single-step exception occurs. The single-step exception is auto vectored to exception number 1. Precisely, exception 1 occurs as a trap after the instruction following the instruction which set TF. In typical practice, a debugger sets the TF bit of a flag register image on the debugger's stack. It then typically transfers control to the user program and loads the flag image with a signal instruction, the IRET instruction. The single-step trap occurs after executing one instruction of the user program.

Because exception 1 occurs as a trap (that is, it occurs after the instruction has already executed), the CS:EIP pushed onto the debugger's stack points to the next unexecuted instruction of the program being debugged. An exception 1 handler, merely by ending with an IRET instruction, can therefore efficiently support single-stepping through a user program.

12.3 Debug Registers

The Debug Registers are an advanced debugging feature of the Military Intel486 processor. They allow data access breakpoints as well as code execution breakpoints. Because the breakpoints are indicated by on-chip registers, an instruction execution breakpoint can be placed in ROM code or in code shared by several tasks, neither of which can be supported by the INT3 breakpoint opcode.

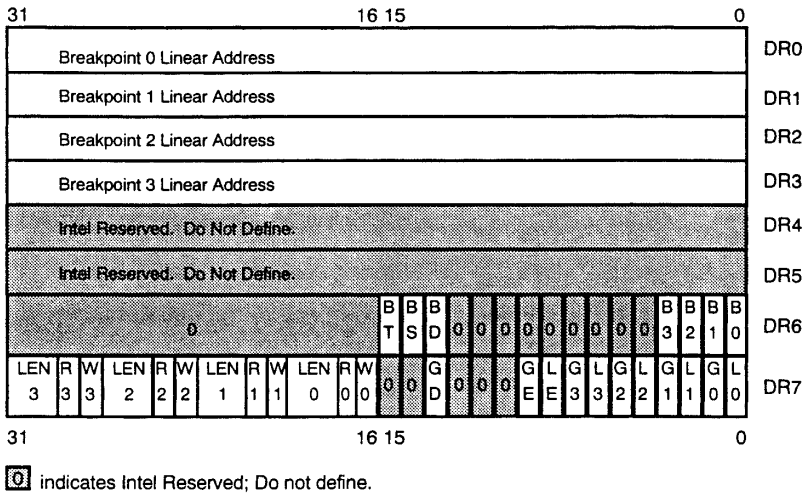
The Military Intel486 processor contains six Debug Registers, providing the ability to specify up to four distinct breakpoints addresses, breakpoint control options, and read breakpoint status. Initially after reset, breakpoints are in the disabled state. Therefore, no breakpoints will occur unless the debug registers are programmed. Breakpoints set up in the Debug Registers are auto vectored to exception number 1.

12.3.1 LINEAR ADDRESS BREAKPOINT REGISTERS (DR0–DR3)

Up to four breakpoint addresses can be specified by writing into Debug Registers DR0–DR3, shown in Figure 12-1. The breakpoint addresses specified are 32-bit linear addresses. Military Intel486 processor hardware continuously compares the linear breakpoint addresses in DR0–DR3 with the linear addresses generated by executing software (a linear address is the result of computing the effective address and adding the 32-bit segment base address). Note that if paging is not enabled the linear address equals the physical address. If paging is enabled, the linear address is translated to a physical 32-bit address by the on-chip paging unit. Regardless of whether paging is enabled or not, however, the breakpoint registers hold linear addresses.

12.3.2 DEBUG CONTROL REGISTER (DR7)

A Debug Control Register, DR7 shown in Figure 12-1, allows several debug control functions such as enabling the breakpoints and setting up other control options for the breakpoints. The fields within the Debug Control Register, DR7, are as follows:



NOTE:
See section 4.2.7

Figure 12-1. Debug Registers

LENi (breakpoint length specification bits)

A 2-bit LEN field exists for each of the four breakpoints. LEN specifies the length of the associated breakpoint field. The choices for data breakpoints are: 1 byte, 2 bytes, and 4 bytes. Instruction execution breakpoints must have a length of 1 (LENi = 00). Encoding of the LENi field is as described in Table 12-1.

The LENi field controls the size of breakpoint field i by controlling whether all low-order linear address bits in the breakpoint address register are used to detect the breakpoint event. Therefore, all breakpoint fields are aligned; 2-byte breakpoint fields begin on Word boundaries, and 4-byte breakpoint fields begin on Dword boundaries.

Figure 12-2 is an example of various size breakpoint fields. Assume the breakpoint linear address in DR2 is 00000005H. In that situation, the Figure 12-2 indicates the region of the breakpoint field for lengths of 1, 2, or 4 bytes.

RWi (memory access qualifier bits)

A 2-bit RW field exists for each of the four breakpoints. The 2-bit RW field specifies the type of usage which must occur in order to activate the associated breakpoint.

Table 12-1. LENi Encoding

LENi Encoding	Breakpoint Field Width	Usage of Least Significant Bits in Breakpoint Address Register i, (i = 0-3)
00	1 byte	All 32-bits used to specify a single-byte breakpoint field.
01	2 bytes	A1–A31 used to specify a two-byte, word-aligned breakpoint field. A0 in Breakpoint Address Register is not used.
10	Undefined—do not use this encoding	
11	4 bytes	A2–A31 used to specify a four-byte, dword-aligned breakpoint field. A0 and A1 in Breakpoint Address Register are not used.

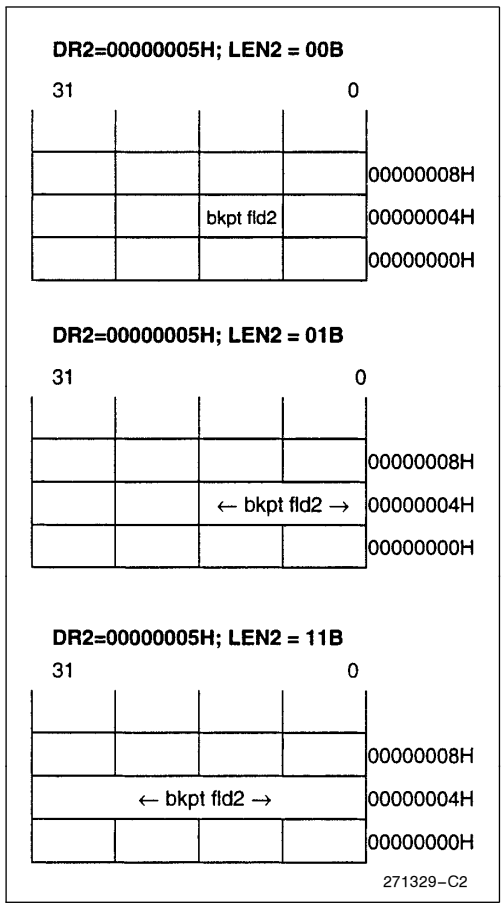


Figure 12-2. Size Breakpoint Fields

Table 12-2. RW Encoding

RW Encoding	Usage Causing Breakpoint
00	Instruction execution only
01	Data writes only
10	Undefined-do not use this encoding
11	Data reads and writes only

RW encoding 00 is used to set up an instruction execution breakpoint. RW encodings 01 or 11 are used to set up write-only or read/write data breakpoints.

Note that **instruction execution breakpoints are taken as faults** (i.e., before the instruction executes), but **data breakpoints are taken as traps** (i.e., after the data transfer takes place).

Using LENi and RWi to Set Data Breakpoint i

A data breakpoint can be set up by writing the linear address into DRi (i = 0–3). For data breakpoints, RWi can = 01 (write-only) or 11 (write/read). LEN can = 00, 01, or 11.

If a data access entirely or partly falls within the data breakpoint field, the data breakpoint condition has occurred, and if the breakpoint is enabled, an exception 1 trap will occur.

Using LENi and RWi to Set Instruction Execution Breakpoint i

An instruction execution breakpoint can be set up by writing address of the beginning of the instruction (including prefixes if any) into DRi (i = 0–3). RWi must = 00 and LEN must = 00 for instruction execution breakpoints.

If the instruction beginning at the breakpoint address is about to be executed, the instruction execution breakpoint condition has occurred, and if the breakpoint is enabled, an exception 1 fault will occur before the instruction is executed.

Note that an instruction execution breakpoint address must be equal to the **beginning** byte address of an instruction (including prefixes) in order for the instruction execution breakpoint to occur.

GD (Global Debug Register access detect)

The Debug Registers can only be accessed in Real Mode or at privilege level 0 in Protected Mode. The GD bit, when set, provides extra protection against **any** Debug Register access even in Real Mode or at privilege level 0 in Protected Mode. This additional protection feature is provided to guarantee that a software debugger can have full control over the Debug Register resources when required. The GD bit, when set, causes an exception 1 fault if an instruction attempts to read or write any Debug Register. The GD bit is then automatically cleared when the exception 1 handler is invoked, allowing the exception 1 handler free access to the debug registers.

GE and LE (Exact data breakpoint match, global and local)

The breakpoint mechanism of the Military Intel486 processor differs from that of the Intel386 processor. The Military Intel486 processor always does exact data breakpoint matching, regardless of GE/LE bit settings. Any data breakpoint trap will be reported exactly after completion of the instruction that caused the operand transfer. Exact reporting is provided by forcing the Military Intel486 processor execution unit to wait for completion of data operand transfers before beginning execution of the next instruction.

When the Military Intel486 processor performs a task switch, the LE bit is cleared. Thus, the LE bit supports fast task switching out of tasks, that have enabled the exact data breakpoint match for their task-local breakpoints. The LE bit is cleared by the Military Intel486 processor during a task switch, to avoid having exact data breakpoint match enabled in the new task. Note that exact data breakpoint match must be re-enabled under software control.

The Military Intel486 processor GE bit is unaffected during a task switch. The GE bit supports exact data breakpoint match that is to remain enabled during all tasks executing in the system.

Note that **instruction execution** breakpoints are always reported exactly.

Gi and Li (breakpoint enable, global and local)

If either Gi or Li is set then the associated breakpoint (as defined by the linear address in DRi, the length in LENi and the usage criteria in RWi) is enabled. If either Gi or Li is set, and the Military Intel486 processor detects the ith breakpoint condition, then the exception 1 handler is invoked.

When the Military Intel486 processor performs a task switch to a new Task State Segment (TSS), all Li bits are cleared. Thus, the Li bits support fast task switching out of tasks that use some task-local breakpoint registers. The Li bits are cleared by the Military Intel486 processor during a task switch, to avoid spurious exceptions in the new task. Note that the breakpoints must be re-enabled under software control.

All Military Intel486 processor Gi bits are unaffected during a task switch. The Gi bits support breakpoints that are active in all tasks executing in the system.

12.3.3 DEBUG STATUS REGISTER (DR6)

A Debug Status Register, DR6 shown in Figure 12-1, allows the exception 1 handler to easily determine why it was invoked. Note the exception 1 handler can be invoked as a result of one of several events:

1. DR0 Breakpoint fault/trap.
2. DR1 Breakpoint fault/trap.
3. XDR2 Breakpoint fault/trap.
4. XDR3 Breakpoint fault/trap.
5. XSingle-step (TF) trap.
6. XTask switch trap.
7. XFault due to attempted debug register access when GD = 1.

The Debug Status Register contains single-bit flags for each of the possible events invoking exception 1. Note below that some of these events are faults (exception taken before the instruction is executed), while other events are traps (exception taken after the debug events occurred).

The flags in DR6 are set by the hardware but never cleared by hardware. Exception 1 handler software should clear DR6 before returning to the user program to avoid future confusion in identifying the source of exception 1.

The fields within the Debug Status Register, DR6, are as follows:

Bi (debug fault/trap due to breakpoint 0–3)

Four breakpoint indicator flags, B0–B3, correspond one-to-one with the breakpoint registers in DR0–DR3. A flag Bi is set when the condition described by DRi, LENi, and RWi occurs.

If Gi or Li is set, and if the ith breakpoint is detected, the Military Intel486 processor will invoke the exception 1 handler. The exception is handled as a fault if an instruction execution breakpoint occurred, or as a trap if a data breakpoint occurred.

IMPORTANT NOTE:

A flag Bi is set whenever the hardware detects a match condition on **enabled** breakpoint i. Whenever a match is detected on at least one **enabled** breakpoint i, the hardware immediately sets all Bi bits corresponding to breakpoint conditions matching at that instant, whether enabled **or not.** Therefore, the exception 1 handler may see that multiple Bi bits are set, but only set Bi bits corresponding to **enabled** breakpoints (Li or Gi set) are **true** indications of why the exception 1 handler was invoked.

BD (debug fault due to attempted register access when GD bit set)

This bit is set if the exception 1 handler was invoked due to an instruction attempting to read or write to the debug registers when GD bit was set. If such an event occurs, then the GD bit is automatically cleared when the exception 1 handler is invoked, allowing handler access to the debug registers.

BS (debug trap due to single-step)

This bit is set if the exception 1 handler was invoked due to the TF bit in the flag register being set (for single-stepping).

BT (debug trap due to task switch)

This bit is set if the exception 1 handler was invoked due to a task switch occurring to a task having a Military Intel486 processor TSS with the T bit set. Note the task switch into the new task occurs normally, but before the first instruction of the task is executed, the exception 1 handler is invoked. With respect to the task switch operation, the operation is considered to be a trap.

12.3.4 USE OF RESUME FLAG (RF) IN FLAG REGISTER

The Resume Flag (RF) in the flag word can suppress an instruction execution breakpoint when the exception 1 handler returns to a user program at a user address which is also an instruction execution breakpoint.

13.0 INSTRUCTION SET SUMMARY

This section describes the Military Intel486 processor instruction set. Detailed information on the CPUID instruction can be found in Appendix A: Feature Determination. Further details of the instruction encoding are then provided in section 13.1, which describes the entire encoding structure and the definition of all fields occurring within the Military Intel486 processor instructions.

13.1 Instruction Encoding

13.1.1 OVERVIEW

All instruction encodings are subsets of the general instruction format shown in Figure 13-1. Instructions consist of one or two primary opcode bytes, possibly an address specifier consisting of the "mod r/m" byte and "scaled index" byte, a displacement if required, and an immediate data field if required.

Within the primary opcode or opcodes, smaller encoding fields may be defined. These fields vary according to the class of operation. The fields define such information as direction of the operation, size of the displacements, register encoding, or sign extension.

Almost all instructions referring to an operand in memory have an addressing mode byte following the primary opcode byte(s). This byte, the mod r/m byte, specifies the address mode to be used. Certain encodings of the mod r/m byte indicate a second addressing byte, the scale-index-base byte, follows the mod r/m byte to fully specify the addressing mode.

Addressing modes can include a displacement immediately following the mod r/m byte, or scaled index byte. If a displacement is present, the possible sizes are 8, 16 or 32 bits.

If the instruction specifies an immediate operand, the immediate operand follows any displacement bytes. The immediate operand, if specified, is always the last field of the instruction.

Figure 13-1 illustrates several of the fields that can appear in an instruction, such as the mod field and the r/m field, but the figure does not show all fields. Several smaller fields also appear in certain instructions, sometimes within the opcode bytes themselves. Table 13-1 is a complete list of all fields appearing in the Military Intel486 processor instruction set. Following Table 13-1 are detailed tables for each field.

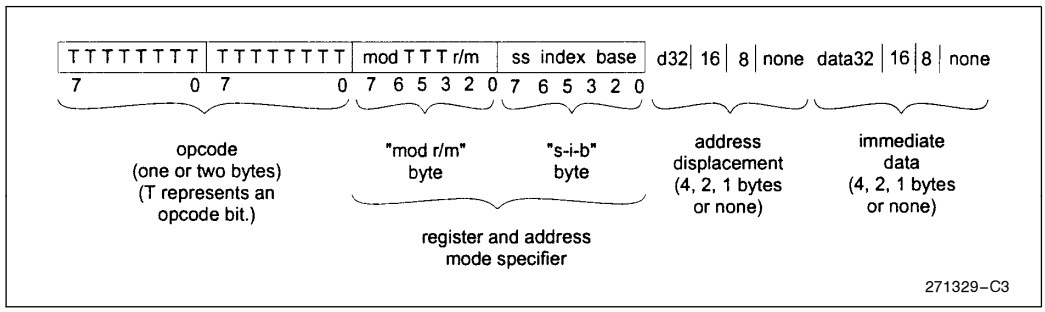


Figure 13-1. General Instruction Format

Table 13-1. Fields within Military Intel486™ Processor Instructions

Field Name	Description	Number of Bits
w	Specifies if Data is Byte or Full Size (Full Size is either 16 or 32 Bits)	1
d	Specifies Direction of Data Operation	1
s	Specifies if an Immediate Data Field Must be Sign-Extended	1
reg	General Register Specifier	3
mod r/m	Address Mode Specifier (Effective Address can be a General Register)	2 for mod; 3 for r/m
ss	Scale Factor for Scaled Index Address Mode	2
index	General Register to be used as Index Register	3
base	General Register to be used as Base Register	3
sreg2	Segment Register Specifier for CS, SS, DS, ES	2
sreg3	Segment Register Specifier for CS, SS, DS, ES, FS, GS	3
tttn	For Conditional Instructions, Specifies a Condition Asserted or a Condition Negated	4

NOTE:
Table 13-15 through Table 13-19 show encoding of individual instructions.

13.1.2 32-BIT EXTENSIONS OF THE INSTRUCTION SET

With the Military Intel486 processor, the 8086/80186/80286 instruction set is extended in two orthogonal directions: 32-bit forms of all 16-bit instructions are added to support the 32-bit data types, and 32-bit addressing modes are made available for all instructions referencing memory. This orthogonal instruction set extension is accomplished having a Default (D) bit in the code segment descriptor, and by having 2 prefixes to the instruction set.

Whether the instruction defaults to operations of 16 bits or 32 bits depends on the setting of the D bit in the code segment descriptor, which gives the default length (either 32 bits or 16 bits) for both operands and effective addresses when executing that code segment. In the Real Address Mode or Virtual 8086 Mode, no code segment descriptors are used, but a D value of 0 is assumed internally by the Military Intel486 processor when operating in those modes (for 16-bit default sizes compatible with the 8086/80186/80286).

Two prefixes, the Operand Size Prefix and the Effective Address Size Prefix, allow overriding individually the Default selection of operand size and effective address size. These prefixes may precede any opcode bytes and affect only the instruction they precede. If necessary, one or both of the prefixes may be placed before the opcode bytes. The presence of the Operand Size Prefix and the Effective Address Prefix will toggle the operand size or the effective address size, respectively, to the value ‘‘opposite’’ from the Default setting. For example, if the default operand size is for 32-bit data operations, then presence of the Operand Size Prefix toggles the instruction to 16-bit data operation. As another example, if the default effective address size is 16 bits, presence of the Effective Address Size prefix toggles the instruction to use 32-bit effective address computations.

These 32-bit extensions are available in all Military Intel486 processor modes, including the Real Address Mode or the Virtual 8086 Mode. In these modes the default is always 16 bits, so prefixes are needed to specify 32-bit operands or addresses. For instructions with more than one prefix, the order of prefixes is unimportant.

Unless specified otherwise, instructions with 8-bit and 16-bit operands do not affect the contents of the high-order bits of the extended registers.

13.1.3 ENCODING OF INTEGER INSTRUCTION FIELDS

Within the instruction are several fields indicating register selection, addressing mode and so on. The exact encodings of these fields are defined immediately ahead.

13.1.3.1 Encoding of Operand Length (w) Field

For any given instruction performing a data operation, the instruction is executing as a 32-bit operation or a 16-bit operation. Within the constraints of the operation size, the w field encodes the operand size as either one byte or the full operation size, as shown in the table below.

Table 13-2. Encoding of Operand Length (w) Field

w Field	Operand Size during 16-Bit Data Operations	Operand Size during 32-Bit Data Operations
0	8 Bits	8 Bits
1	16 Bits	32 Bits

13.1.3.2 Encoding of the General Register (reg) Field

The general register is specified by the reg field, which may appear in the primary opcode bytes, or as the reg field of the ''mod r/m'' byte, or as the r/m field of the ''mod r/m'' byte.

Table 13-3. Encoding of reg Field when the w Field Is Not Present in Instruction

reg Field	Register Selected during 16-Bit Data Operations	Register Selected during 32-Bit Data Operations
000	AX	EAX
001	CX	ECX
010	DX	EDX
011	BX	EBX
100	SP	ESP
101	BP	EBP
110	SI	ESI
111	DI	EDI

Table 13-4. Encoding of reg Field when the w Field Is Present in Instruction

Register Specified by reg Field during 16-Bit Data Operations:		
reg	Function of w Field	
	(when w = 0)	(when w = 1)
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI
Register Specified by reg Field during 32-Bit Data Operations		
reg	Function of w Field	
	(when w = 0)	(when w = 1)
000	AL	EAX
001	CL	ECX
010	DL	EDX
011	BL	EBX
100	AH	ESP
101	CH	EBP
110	DH	ESI
111	BH	EDI

13.1.3.3 Encoding of the Segment Register (sreg) Field

The sreg field in certain instructions is a 2-bit field allowing one of the four 80286 segment registers to be specified. The sreg field in other instructions is a 3-bit field, allowing the Military Intel486 processor FS and GS segment registers to be specified.

Table 13-5. 2-Bit sreg2 Field

2-bit sreg2 Field	Segment Register Selected
00	ES
01	CS
10	SS
11	DS

Table 13-6. 3-Bit sreg3 Field

3-bit sreg3 Field	Segment Register Selected
000	ES
001	CS
010	SS
011	DS
100	FS
101	GS
110	do not use
111	do not use

13.1.3.4 Encoding of Address Mode

Except for special instructions, such as PUSH or POP, where the addressing mode is pre-determined, the addressing mode for the current instruction is specified by addressing bytes following the primary opcode. The primary addressing byte is the ''mod r/m'' byte, and a second byte of addressing information, the ''s-i-b'' (scale-index-base) byte, can be specified.

The s-i-b byte (scale-index-base byte) is specified when using 32-bit addressing mode and the ''mod r/m'' byte has r/m = 100 and mod = 00, 01 or 10. When the sib byte is present, the 32-bit addressing mode is a function of the mod, ss, index, and base fields.

The primary addressing byte, the ''mod r/m'' byte, also contains three bits (shown as TTT in Figure 13-1) sometimes used as an extension of the primary opcode. The three bits, however, may also be used as a register field (reg).

When calculating an effective address, either 16-bit addressing or 32-bit addressing is used. 16-bit addressing uses 16-bit address components to calculate the effective address while 32-bit addressing uses 32-bit address components to calculate the effective address. When 16-bit addressing is used, the ''mod r/m'' byte is interpreted as a 16-bit addressing mode specifier. When 32-bit addressing is used, the ''mod r/m'' byte is interpreted as a 32-bit addressing mode specifier.

Tables 13-7, 13-8, and 13-9 define all encodings of all 16-bit addressing modes and 32-bit addressing modes.

Table 13-7. Encoding of 16-Bit Address Mode with "mod r/m" Byte

mod r/m	Effective Address
00 000	DS:[BX + SI]
00 001	DS:[BX + DI]
00 010	SS:[BP + SI]
00 011	SS:[BP + DI]
00 100	DS:[SI]
00 101	DS:[DI]
00 110	DS:d16
00 111	DS:[BX]
01 000	DS:[BX + SI + d8]
01 001	DS:[BX + DI + d8]
01 010	SS:[BP + SI + d8]
01 011	SS:[BP + DI + d8]
01 100	DS:[SI + d8]
01 101	DS:[DI + d8]
01 110	SS:[BP + d8]
01 111	DS:[BX + d8]

mod r/m	Effective Address
10 000	DS:[BX + SI + d16]
10 001	DS:[BX + DI + d16]
10 010	SS:[BP + SI + d16]
10 011	SS:[BP + DI + d16]
10 100	DS:[SI + d16]
10 101	DS:[DI + d16]
10 110	SS:[BP + d16]
10 111	DS:[BX + d16]
11 000	register—see below
11 001	register—see below
11 010	register—see below
11 011	register—see below
11 100	register—see below
11 101	register—see below
11 110	register—see below
11 111	register—see below

Register Specified by r/m during 16-Bit Data Operations

mod r/m	Function of w Field	
	(when w = 0)	(when w = 1)
11 000	AL	AX
11 001	CL	CX
11 010	DL	DX
11 011	BL	BX
11 100	AH	SP
11 101	CH	BP
11 110	DH	SI
11 111	BH	DI

Register Specified by r/m during 32-Bit Data Operations

mod r/m	Function of w Field	
	(when w = 0)	(when w = 1)
11 000	AL	EAX
11 001	CL	ECX
11 010	DL	EDX
11 011	BL	EBX
11 100	AH	ESP
11 101	CH	EBP
11 110	DH	ESI
11 111	BH	EDI

Table 13-8. Encoding of 32-Bit Address Mode with "mod r/m" Byte (No "s-i-b" Byte Present)

mod r/m	Effective Address
00 000	DS:[EAX]
00 001	DS:[ECX]
00 010	DS:[EDX]
00 011	DS:[EBX]
00 100	s-i-b is present
00 101	DS:d32
00 110	DS:[ESI]
00 111	DS:[EDI]
01 000	DS:[EAX + d8]
01 001	DS:[ECX + d8]
01 010	DS:[EDX + d8]
01 011	DS:[EBX + d8]
01 100	s-i-b is present
01 101	SS:[EBP + d8]
01 110	DS:[ESI + d8]
01 111	DS:[EDI + d8]

mod r/m	Effective Address
10 000	DS:[EAX + d32]
10 001	DS:[ECX + d32]
10 010	DS:[EDX + d32]
10 011	DS:[EBX + d32]
10 100	s-i-b is present
10 101	SS:[EBP + d32]
10 110	DS:[ESI + d32]
10 111	DS:[EDI + d32]
11 000	register—see below
11 001	register—see below
11 010	register—see below
11 011	register—see below
11 100	register—see below
11 101	register—see below
11 110	register—see below
11 111	register—see below

Register Specified by reg or r/m during 16-Bit Data Operations:

mod r/m	Function of w Field	
	(when w = 0)	(when w = 1)
11 000	AL	AX
11 001	CL	CX
11 010	DL	DX
11 011	BL	BX
11 100	AH	SP
11 101	CH	BP
11 110	DH	SI
11 111	BH	DI

Register Specified by reg or r/m during 32-Bit Data Operations:

mod r/m	Function of w Field	
	(when w = 0)	(when w = 1)
11 000	AL	EAX
11 001	CL	ECX
11 010	DL	EDX
11 011	BL	EBX
11 100	AH	ESP
11 101	CH	EBP
11 110	DH	ESI
11 111	BH	EDI

Table 13-9. Encoding of 32-Bit Address Mode (“mod r/m” Byte and “s-i-b” Byte Present)

mod base	Effective Address
00 000	DS:[EAX + (scaled index)]
00 001	DS:[ECX + (scaled index)]
00 010	DS:[EDX + (scaled index)]
00 011	DS:[EBX + (scaled index)]
00 100	SS:[ESP + (scaled index)]
00 101	DS:[d32 + (scaled index)]
00 110	DS:[ESI + (scaled index)]
00 111	DS:[EDI + (scaled index)]
01 000	DS:[EAX + (scaled index) + d8]
01 001	DS:[ECX + (scaled index) + d8]
01 010	DS:[EDX + (scaled index) + d8]
01 011	DS:[EBX + (scaled index) + d8]
01 100	SS:[ESP + (scaled index) + d8]
01 101	SS:[EBP + (scaled index) + d8]
01 110	DS:[ESI + (scaled index) + d8]
01 111	DS:[EDI + (scaled index) + d8]
10 000	DS:[EAX + (scaled index) + d32]
10 001	DS:[ECX + (scaled index) + d32]
10 010	DS:[EDX + (scaled index) + d32]
10 011	DS:[EBX + (scaled index) + d32]
10 100	SS:[ESP + (scaled index) + d32]
10 101	SS:[EBP + (scaled index) + d32]
10 110	DS:[ESI + (scaled index) + d32]
10 111	DS:[EDI + (scaled index) + d32]

NOTE:
Mod field in “mod r/m” byte; ss, index, base fields in “s-i-b” byte.

ss	Scale Factor
00	x1
01	x2
10	x4
11	x8
Index	**Index Register**
000	EAX
001	ECX
010	EDX
011	EBX
100	no index reg**
101	EBP
110	ESI
111	EDI

****IMPORTANT NOTE:**
When index field is 100, indicating “no index register,” then ss field MUST equal 00. If index is 100 and ss does not equal 00, the effective address is undefined.

13.1.3.5 Encoding of Operation Direction (d) Field

In many two-operand instructions the d field is present to indicate which operand is considered the source and which is the destination.

Table 13-10. Encoding of Operation Direction (d) Field

d	Direction of Operation
0	Register/Memory ← Register ''reg'' Field Indicates Source Operand; ''mod r/m'' or ''mod ss index base'' Indicates Destination Operand
1	Register ← Register/Memory ''reg'' Field Indicates Destination Operand; ''mod r/m'' or ''mod ss index base'' Indicates Source Operand

13.1.3.6 Encoding of Sign-Extend (s) Field

The s field occurs primarily to instructions with immediate data fields. The s field has an effect only if the size of the immediate data is 8 bits and is being placed in a 16-bit or 32-bit destination.

Table 13-11. Encoding of Sign-Extend (s) Field

S	Effect on Immediate Data 8	Effect on Immediate Data 16|32
0	None	None
1	Sign-Extend Data 8 to Fill 16-bit or 32-bit Destination	None

13.1.3.7 Encoding of Conditional Test (tttn) Field

For the conditional instructions (conditional jumps and set on condition), tttn is encoded with n indicating to use the condition (n = 0) or its negation (n = 1), and ttt giving the condition to test.

Table 13-12. Encoding of Conditional Test (tttn) Field

Mnemonic	Condition	tttn
O	Overflow	0000
NO	No Overflow	0001
B/NAE	Below/Not Above or Equal	0010
NB/AE	Not Below/Above or Equal	0011
E/Z	Equal/Zero	0100
NE/NZ	Not Equal/Not Zero	0101
BE/NA	Below or Equal/Not Above	0110
NBE/A	Not Below or Equal/Above	0111
S	Sign	1000
NS	Not Sign	1001
P/PE	Parity/Parity Even	1010
NP/PO	Not Parity/Parity Odd	1011
L/NGE	Less Than/Not Greater or Equal	1100
NL/GE	Not Less Than/Greater or Equal	1101
LE/NG	Less Than or Equal/Greater Than	1110
NLE/G	Not Less or Equal/Greater Than	1111

13.1.3.8 Encoding of Control or Debug or Test Register (eee) Field

For the loading and storing of the Control, Debug and Test registers.

Table 13-13. Encoding of Control or Debug or Test Register (eee) Field

eee Code	Reg Name
When Interpreted as Control Register Field:	
000	CR0
010	CR2
011	CR3
When Interpreted as Debug Register Field:	
000	DR0
001	DR1
010	DR2
011	DR3
110	DR6
111	DR7
When Interpreted as Test Register Field:	
011	TR3
100	TR4
101	TR5
110	TR6
111	TR7

Do not use any other encoding

Table 13-14. Encoding of Floating-Point Instruction Fields

	Instruction								Optional	
	First Byte				Second Byte				Fields	
1	11011	OPA		1	mod		1 OPB	r/m	s-i-b	disp
2	11011	MF		OPA	mod		OPB	r/m	s-i-b	disp
3	11011	d	P	OPA	1	1	OPB	ST(i)		
4	11011	0	0	1	1	1	1	OP		
5	11011	0	1	1	1	1	1	OP		
	15–11	10	9	8	7	6	5	4 3 2 1 0		

13.1.4 ENCODING OF FLOATING POINT INSTRUCTION FIELDS

Instructions for the FPU assume one of the five forms shown in the following table. In all cases, instructions are at least two bytes long and begin with the bit pattern 11011B.

OP = Instruction opcode, possible split into two fields OPA and OPB

MF = Memory Format
00–32-bit real
01–32-bit integer
10–64-bit real
11–16-bit integer

P = Pop
0–Do not pop stack
1–Pop stack after operation

d = Destination
0–Destination is ST(0)
1–Destination is ST(i)

R XOR d = 0–Destination (op) Source

R XOR d = 1–Source (op) Destination

ST(i) = Register stack element *i*
000 = Stack top
001 = Second stack element
111 = Eighth stack element

mod (Mode field) and r/m (Register/Memory specifier) have the same interpretation as the corresponding fields of the integer instructions.

s-i-b (Scale Index Base) byte and disp (displacement) are optionally present in instructions that have mod and r/m fields. Their presence depends on the values of mod and r/m, as for integer instructions.

13.2 Clock Count Summary

To calculate elapsed time for an instruction, multiply the instruction clock count, as listed in Table 13-15 through Table 13-19 by the processor core clock period (e.g., 10 ns for a 100-MHz IntelDX4 processor).

13.2.1 INSTRUCTION CLOCK COUNT ASSUMPTIONS

The Military Intel486 processor instruction core clock count tables give clock counts assuming data and instruction accesses hit in the cache. The combined instruction and data cache hit rate is over 90%.

A cache miss will force the Military Intel486 processor to run an external bus cycle. The Military Intel486 processor 32-bit burst bus is defined as r-b-w.

Where:

r = The number of bus clocks in the first cycle of a burst read or the number of clocks per data cycle in a non-burst read.

b = The number of bus clocks for the second and subsequent cycles in a burst read.

w = The number of bus clocks for a write.

The clock counts in the cache miss penalty column assume a 2-1-2 bus. For slower buses add r-2 clocks to the cache miss penalty for the first dword accessed. Other factors also affect instruction clock counts.

Instruction Clock Count Assumptions

1. The external bus is available for reads or writes at all times. Else add bus clocks to reads until the bus is available.
2. Accesses are aligned. Add three core clocks to each misaligned access.
3. Cache fills complete before subsequent accesses to the same line. If a read misses the cache during a cache fill due to a previous read or prefetch, the read must wait for the cache fill to complete. If a read or write accesses a cache line still being filled, it must wait for the fill to complete.
4. If an effective address is calculated, the base register is not the destination register of the preceding instruction. If the base register is the destination register of the preceding instruction add 1 to the core clock counts shown. Back-to-back PUSH and POP instructions are not affected by this rule.
5. An effective address calculation uses one base register and does not use an index register. However, if the effective address calculation uses an index register, 1 core clock **may** be added to the clock count shown.
6. The target of a jump is in the cache. If not, add r clocks for accessing the destination instruction of a jump. If the destination instruction is not completely contained in the first dword read, add a maximum of 3b bus clocks. If the destination instruction is not completely contained in the first 16 byte burst, add a maximum of another r + 3b bus clocks.
7. If no write buffer delay, w bus clocks are added only in the case in which all write buffers are full.
8. Displacement and immediate not used together. If displacement and immediate used together, 1 core clock **may** be added to the core clock count shown.
9. No invalidate cycles. Add a delay of 1 bus clock for each invalidate cycle if the invalidate cycle contends for the internal cache/external bus when the Military Intel486 processor needs to use it.
10. Page translation hits in TLB. A TLB miss will add 13, 21 or 28 bus clocks + 1 possible core clock to the instruction depending on whether the Accessed and/or Dirty bit in neither, one or both of the page entries needs to be set in memory. This assumes that neither page entry is in the data cache and a page fault does not occur on the address translation.
11. No exceptions are detected during instruction execution. Refer to Interrupt core Clock Counts Table for extra clocks if an interrupt is detected.
12. Instructions that read multiple consecutive data items (i.e. task switch, POPA, etc.) and miss the cache are assumed to start the first access on a 16-byte boundary. If not, an extra cache line fill may be necessary which may add up to (r + 3b) bus clocks to the cache miss penalty.

Table 13-15. Clock Count Summary

Instruction	Format	Cache Hit	Penalty if Cache Miss	Notes
INTEGER OPERATIONS				
MOV = Move:				
reg1 to reg2	1000 100w : 11 reg1 reg2	1		
reg2 to reg1	1000 101w : 11 reg1 reg2	1		
memory to reg	1000 100w : mod reg r/m	1	2	
Immediate to reg	1100 011w : 11000 reg : immediate data	1		
or	1011W reg : immediate data	1		
Immediate to Memory	1100 01w : mod 000 r/m : displacement immediate	1		
Memory to Accumulator	1010 000w : full displacement	1	2	
Accumulator to Memory	1010 001w : full displacement	1		
MOVSX/MOVZX = Move with Sign/Zero Extension				
reg2 to reg1	0000 1111 : 1011 z11w : 11 reg1 reg2	3		
memory to reg	0000 1111 : 1011 z11w : mod reg r/m	3	2	
z instruction 0 MOVZX 1 MOVSX				
PUSH = Push				
reg	1111 1111 : 11 110 reg	4		
or	01010 reg	1		
memory	1111 1111 : mod 110 r/m	4	1	1
immediate	0110 10s0 : immediate data	1		
PUSHA = Push All	0110 0000	11		
POP = Pop				
reg	1000 1111 : 11 000 reg	4	1	
or	01011 reg	1	2	
memory	1000 1111 : mod 000 r/m	5	2	1
POPA = Pop All	0110 0001	9	7/15	16/32
XCHG = Exchange				
reg1 with reg2	1000 011w : 11 reg1 reg2	3		2
Accumulator with reg	10010 reg	3		2
Memory with reg	1000 011w : mod reg r/m	5		2

Table 13-15. Clock Count Summary (Continued)

Instruction	Format	Cache Hit	Penalty if Cache Miss	Notes
INTEGER OPERATIONS (Continued)				
NOP = No Operation	1001 0000	1		
LEA = Load EA to Register	1000 1101 : mod reg r/m			
no index register		1		
with index register		2		
Instruction	TTT			
ADD = Add	000			
ADC = Add with Carry	010			
AND = Logical AND	100			
OR = Logical OR	001			
SUB = Subtract	101			
SBB = Subtract with Borrow	011			
XOR = Logical Exclusive OR	110			
reg1 to reg2	00TT T00w : 11 reg1 reg2	1		
reg2 to reg1	00TT T01w : 11 reg1 reg2	1		
memory to register	00TT T01w : mod reg r/m	2	2	
register to memory	00TT T00w : mod reg r/m	3	6/2	U/L
immediate to register	1000 00sw : 11 TTT reg : immediate register	1		
immediate to Accumulator	00TT T10w : immediate data	1		
immediate to memory	1000 00sw : mod TTT r/m : immediate data	3	6/2	U/L
Instruction	TTT			
INC = Increment	000			
DEC = Decrement	001			
reg	1111 111w : 11 TTT reg	1		
or	01TTT reg	1		
memory	1111 111w : mod TTT r/m	3	6/2	U/L
Instruction	TTT			
NOT = Logical Complement	010			
NEG = Negate	011			
reg	1111 011w : 11 TTT reg	1		
memory	1111 011w : mod TTT r/m	3	6/2	U/L

Table 13-15. Clock Count Summary (Continued)

Instruction	Format	Cache Hit	Penalty if Cache Miss	Notes
INTEGER OPERATIONS (Continued)				
CMP = Compare				
reg1 with reg2	0011 100w : 11 reg1 reg2	1		
reg2 with reg1	0011 101w : 11 reg1 reg2	1		
memory with register	0011 100w : mod reg r/m	2	2	
register with memory	0011 101w : mod reg r/m	2	2	
immediate with register	1000 00sw : 11 111 reg : immediate data	1		
immediate with acc.	0011 110w : immediate data	1		
immediate with memory	1000 00sw : mod 111 r/m : immediate data	2	2	
TEST = Logical Compare				
reg1 and reg2	1000 010w : 11 reg1 reg2	1		
memory and register	1000 010w : mod reg r/m	2	2	
immediate and register	1111 011w : 11 000 reg : immediate data	1		
immediate and acc.	1010100w : immediate data	1		
immediate and memory	1111 011w : mod 000 r/m : immediate data	2	2	
MUL = Multiply (unsigned)				
acc. with register	1111 011w : 11 100 reg			
Multiplier-Byte		13/18		MN/MX,3
Word		13/26		MN/MX,3
Dword		13/42		MN/MX,3
acc. with memory	1111 011w : mod 100 r/m			
Multiplier-Byte		13/18	1	MN/MX,3
Word		13/26	1	MN/MX,3
Dword		13/42	1	MN/MX,3
IMUL = Integer Multiply (unsigned)				
acc. with register	1111 011w : 11 101 reg			
Multiplier-Byte		13/18		MN/MX,3
Word		13/26		MN/MX,3
Dword		13/42		MN/MX,3
acc. with memory	1111 011w : mod 101 r/m			
Multiplier-Byte		13/18		MN/MX,3
Word		13/26		MN/MX,3
Dword		13/42		MN/MX,3
reg1 with reg2	0000 1111 : 10101111 : 11 reg1 reg2			
Multiplier-Byte		13/18		MN/MX,3
Word		13/26		MN/MX,3
Dword		13/42		MN/MX,3

Table 13-15. Clock Count Summary (Continued)

Instruction	Format	Cache Hit	Penalty if Cache Miss	Notes
INTEGER OPERATIONS (Continued)				
IMUL = Integer Multiply (unsigned), (Continued)				
register with memory	0000 1111 : 10101111 : mod reg r/m			
Multiplier-Byte		13/18	1	MN/MX,3
Word		13/26	1	MN/MX,3
Dword		13/42	1	MN/MX,3
reg1 with imm. to reg2	0110 10s1 : 11 reg1 reg2 : immediate data			
Multiplier-Byte		13/18		MN/MX,3
Word		13/26		MN/MX,3
Dword		13/42		MN/MX,3
mem. with imm. to reg.	0110 10s1 : mod reg r/m : immediate data			
Multiplier-Byte		13/18		MN/MX,3
Word		13/26		MN/MX,3
Dword		13/42		MN/MX,3
For the IntelDX4™ Processor Only: **IMUL = Integer Multiply (signed)**				
acc. with register	1111 011w : 11101 reg			
Multiplier-Byte		5/5		MN/MX,3
Word		5/6		MN/MX,3
Dword		6/12		MN/MX,3
acc. with memory	1111 011w : mod 101 r/m			
Multiplier-Byte		5/5		MN/MX,3
Word		5/6		MN/MX,3
Dword		6/12		MN/MX,3
reg1 with reg2	0000 1111 : 1010 1111 : 11 reg1 reg2			
Multiplier-Byte		5/5		MN/MX,3
Word		5/6		MN/MX,3
Dword		6/12		MN/MX,
register with memory	0000 1111 : 1010 1111 : mod reg r/m			
Multiplier-Byte		5/5		MN/MX,3
Word		5/6		MN/MX,3
Dword		6/12		MN/MX,
reg1 with imm. to reg2	0110 10s1 : 11 reg1 reg2 : immediate data			
Multiplier-Byte		5/5		MN/MX,3
Word		5/6		MN/MX,3
Dword		6/12		MN/MX,
mem. with imm. to reg.	0110 10s1 : mod reg r/m : immediate data			
Multiplier-Byte		5/5		MN/MX,3
Word		5/6		MN/MX,3
Dword		6/12		MN/MX,3

Table 13-15. Clock Count Summary (Continued)

Instruction	Format	Cache Hit	Penalty if Cache Miss	Notes
INTEGER OPERATIONS (Continued)				
DIV = Divide (unsigned)				
acc. by register	1111 011w : 11110 reg			
Divisor-Byte		16		
Word		24		
Dword		40		
acc. by memory	1111 011w : mod 110 r/m			
Divisor-Byte		16		
Word		24		
Dword		40		
IDIV = Integer Divide (signed)				
acc. by register	1111 011w : 11111 reg			
Divisor-Byte		19		
Word		27		
Dword		43		
acc. by memory	1111 011w : mod 111 r/m			
Divisor-Byte		20		
Word		28		
Dword		44		
CBW = Convert Byte to Word	1001 1000	3		
CWD = Convert Word to Dword	1001 1001	3		
Instruction	TTT			
ROL = Rotate Left	000			
ROR = Rotate Right	001			
RCL = Rotate Through Carry Left	010			
RDR = Rotate Through Carry Right	011			
SHL/SAL = Shift Logical/ Arithmetic Left	100			
SHR = Shift Logical Right	101			
SAR = Shift Arithmetic Right	111			
Not Through Carry (ROL, ROR, SAR, SHL, and SHR)				
reg by 1	1101 000w : 11 TTT reg	3		
memory by 1	1101 000w : mod TTT r/m	4	6	
reg by CL	1101 001w : 11 TTT reg	3		
memory by CL	1101 001w : mod TTT r/m	4	6	
reg by immediate count	1100 000w : 11 TTT reg : imm. 8-bit data	2		
mem by immediate count	1100 000w : mod TTT r/m : imm. 8-bit data	4	6	

Table 13-15. Clock Count Summary (Continued)

Instruction	Format	Cache Hit	Penalty if Cache Miss	Notes
INTEGER OPERATIONS (Continued)				
Through Carry (RCL and RCR)				
reg by 1	1101 000w : 11 TTT reg	3		
memory by 1	1101 000w : mod TTT r/m	4	6	
reg by CL	1101 001w : 11 TTT reg	8/30		MN/MX,4
memory by CL	1101 001w : mod TTT r/m	9/31		MN/MX,5
reg by immediate count	1100 000w : 11 TTT reg : imm. 8-bit data	8/30		MN/MX,4
mem by immediate count	1100 000w : mod TTT r/m : imm. 8-bit data	9/31		MN/MX,5
Instruction SHLD = Shift Left Double SHRD = Shift Right Double	TTT 100 101			
register with immediate	0000 1111 : 10TT T100 : 11 reg2 reg1 : imm. 8-bit data	2		
memory with immediate	0000 1111 : 10TT T100 : mod reg r/m : imm. 8-bit data	3	6	
register by CL	0000 1111 : 10TT T101 : 11 reg2 reg1	3		
memory by CL	0000 1111 : 10TT T101 : mod reg r/m	4	5	
BSWAP = Byte Swap	0000 1111 : 11001 reg	1		
XADD = Exchange and Add				
reg1, reg2	0000 1111 : 1100 000w : 11 reg2 reg1	3		
memory, reg	0000 1111 : 1100 000w : mod reg r/m	4	6/2	U/L
CMPXCHG = Compare and Exchange				
reg1, reg2	0000 1111 : 1011 000w : 11 reg2 reg1	6		
memory, reg	0000 1111 : 1011 000w : mod reg r/m	7/10	2	6
CONTROL TRANSFER (within segment)				
Note: Times are jump taken/not taken **J_{CCCC} = Jump on cccc**				
8-bit displacement	0111 tttn : 8-bit disp.	3/1		T/NT,23
full displacement	0000 1111 : 1000 tttn : full displacement	3/1		T/NT,23
Note: Times are jump taken/not taken **SET_{CCCC} = Set Byte on cccc (Times are cccc true/false)**				
reg	0000 1111 : 1001 tttn : 11 000 reg	4/3		
memory	0000 1111 : 1001 tttn : mod 0000 r/m	3/4		

Table 13-15. Clock Count Summary (Continued)

Instruction	Format	Cache Hit	Penalty if Cache Miss	Notes
CONTROL TRANSFER (within segment) (Continued)				
Mnemonic cccc / Condition / tttn				
O — Overflow — 0000				
NO — No Overflow — 0001				
B/NAE — Below/Not Above or Equal — 0010				
NB/AE — Not Below/Above or Equal — 0011				
E/Z — Equal Zero — 0100				
NE/NZ — Not Equal/Not Zero — 0101				
BE/NA — Below or Equal/Not Above — 0110				
NBE/A — Not Below or Equal/Above — 0111				
S — Sign — 1000				
NS — Not Sign — 1001				
P/PE — Parity/Parity Even — 1010				
NP/PO — Not Parity/Parity Odd — 1011				
L/NGE — Less Than/Not Greater or Equal — 1100				
NL/GE — Not Less Than/Greater or Equal — 1101				
LE/NG — Less Than or Equal/Greater Than — 1110				
NLE/G — Not Less Than or Equal/Greater Than — 1111				
LOOP = LOOP CX Times	1110 0010 : 8-bit disp.	7/6		L/NL,23
LOOPZ/LOOPE = Loop with Zero/Equal	1110 0001 : 8-bit disp.	9/6		L/NL,23
LOOPNZ/LOOPNE = Loop While Not Zero	1110 0000 : 8-bit disp.	9/6		L/NL,23
JCXZ = Jump on CX Zero	1110 0011 : 8-bit disp.	8/5		T/NT,23
JECXZ = Jump on ECX Zero (Address Size Prefix Differentiates JCXZ for JECXZ)	1110 0011 : 8-bit disp.	8/5		T/NT,23
JMP = Unconditional Jump (within segment)				
Short	1110 1011 : 8-bit disp.	3		7,23
Direct	1110 1001 : full displacement	3		7,23
Register Indirect	1111 1111 : 11 100 reg	5		7,23
Memory Indirect	1111 1111 : mod 100 r/m	5	5	7
CALL = Call (within segment)				
Direct	1110 1000 : full displacement	3		7,23
Register Indirect	1111 1111 : 11 010 reg	5		7,23
Memory Indirect	1111 1111 : mod 010 reg	5	5	7
RET = Return from CALL (within segment)	1100 0011	5	5	
Adding Immediate to SP	1100 0010 : 16-bit disp.	5	5	
ENTER = Enter Procedure	1100 1000 : 16-bit disp., 8-bit level			
Level = 0		14		
Level = 1		17		
Level (L) > 1		17+3L		8
LEAVE = Leave Procedure	1100 1001	5	1	

Table 13-15. Clock Count Summary (Continued)

Instruction	Format	Cache Hit	Penalty if Cache Miss	Notes
MULTIPLE-SEGMENT INSTRUCTIONS				
MOV = Move				
reg. to segment reg.	1000 1110 : 11 sreg3 reg	3/9	0/3	RV/P,9
memory to segment reg.	1000 1110 : mod sreg3 r/m	3/9	2/5	RV/P,9
segment reg. to reg.	1000 1100 : 11 sreg3 reg	3		
segment reg. to memory	1000 1100 : mod sreg3 r/m	3		
PUSH = Push				
segment reg. (ES, CS, SS, or DS)	000sreg 2110	3		
segment reg. (FS or GS)	0000 1111 : 10 sreg3001	3		
POP = Pop				
segment reg. (ES, CS, SS, or DS)	000sreg 2111	3/0	2/5	RV/P,9
segment reg. (FS or GS)	0000 1111 : 10 sreg3001	3/9	2/5	RV/P,9
LDS = Load Pointer to DS	1100 0101 : mod reg r/m	6/12	7/10	RV/P,9
LES = Load Pointer to ES	1100 0100 : mod reg r/m	6/12	7/10	RV/P,9
LFS = Load Pointer to FS	0000 1111 : 1011 0100 : mod reg r/m	6/12	7/10	RV/P,9
LGS = Load Pointer to GS	0000 1111 : 1011 0101 : mod reg r/m	6/12	7/10	RV/P,9
LSS = Load Pointer to SS	0000 1111 : 1011 0010 : mod reg r/m	6/12	7/10	RV/P,9
CALL = Call				
Direct intersegment	1001 1010 : unsigned full offset, selector	18	2	R,7,22
to same level		20	3	P,9
thru Gate to same level		35	6	P,9
to inner level, no parameters		69	17	P,9
to inner level, x parameters (d) words		77+4X	17+n	P,11,9
to TSS		37+TS	3	P,10,9
thru Task Gate		38+TS	3	P,10,9
Indirect intersegment	1111 1111 : mod 011 r/m	17	8	R,7
to same level		20	10	P,9
thru Gate to same level		35	13	P,9
to inner level, no parameters		69	24	P,9
to inner level, x parameters (d) words		77+4X	24+n	P,11,9
to TSS		37+TS	10	P,10,9
thru Task Gate		38+TS	10	P,10,9
RET = Return from CALL				
intersegment	1100 1010	13	8	R,7
to same level		17	9	P,9
to outet lever		35	12	P,9
intersegment adding imm. to SP	1100 1010 : 16-bit disp.	14	8	R,7
to same level		18	9	P,9
to outer level		36	12	P,9

Table 13-15. Clock Count Summary (Continued)

Instruction	Format	Cache Hit	Penalty if Cache Miss	Notes
MULTIPLE-SEGMENT INSTRUCTIONS (Continued)				
JMP = Unconditional Jump				
Direct intersegment	1110 1010 : unsigned full offset, selector	17	2	R,7,22
to same level		19	3	P,9
thru Call Gate to same level		32	6	P,9
thru TSS		42+TS	3	P,10,9
thru Task Gate		43+TS	3	P,10,9
Indirect intersegment	1111 1111 : mod 011 r/m	13	9	R,7,9
to same level		18	10	P,9
thru Call Gate to same level		31	13	P,9
thru TSS		41+TS	10	P,10,9
thru Task Gate		42+TS	10	P,10,9
BIT MANIPULATION				
BT = Test Bit				
register, immediate	0000 1111 : 1011 1010 : 11 100 reg : imm. 8-bit data	3		
memory, immediate	0000 1111 : 1011 1010 : mod 100 r/m : imm. 8-bit data	3	1	
reg1, reg2	0000 1111 : 1010 0011 : 11 reg2 reg1	3		
memory, reg	0000 1111 : 1010 0011 : mod reg r/m	8	2	
Instruction	TTT			
BTS = Test Bit and Set	101			
BTR = Test Bit and Reset	110			
BTC = Test Bit and Compliment	111			
register, immediate	0000 1111 : 1011 1010 : 11 TTT reg imm. 8-bit data	6		
memory, immediate	0000 1111 : 1011 1010 : mod TTT r/m imm. 8-bit data	8		U/L
reg1, reg2	0000 1111 : 10TT T011 : 1 1 reg2 reg1	6		
memory, reg	0000 1111 : 10TT T011 : mod reg r/m	13		U/L
BSF = Scan Bit Forward				
reg1, reg2	0000 1111 : 1011 1100 : 11 reg2 reg1	6/42		MN/MX, 12
memory, reg	0000 1111 : 1011 1100 : mod reg r/m	7/43	2	MN/MX, 15
BSR = Scan Bit Reverse				
reg1, reg2	0000 1111 : 1011 1101 : 11 reg2 reg1	6/103		MN/MX, 14
memory, reg	0000 1111 : 1011 1101 : mod reg r/m	7/104	1	MN/MX, 15

Table 13-15. Clock Count Summary (Continued)

Instruction	Format	Cache Hit	Penalty if Cache Miss	Notes
STRING INSTRUCTIONS				
CMPS = **Compare Byte Word**	1010 011w	8	6	16
LODS = **Load Byte/Word** to AL/AX/EAX	1010 111w	5	2	
MOVS = **Move Byte/Word**	1010 010w	7	2	16
SCAS = **Scan Byte/Word**	1010 111w	6	2	
STOS = **Store Byte/Word** from AL/AX/EX	1010 101w	5		
XLAT = **Translate String**	1101 0111	4	2	
REPEATED STRING INSTRUCTIONS Repeated by Count in CX or ECX (C = Count in CX or ECX)				
REPE CMPS = **Compare String** (Find Non-match)	1111 0011 : 1010 011w			
C = 0		5		16, 17
C > 0		7+7c		
REPNE CMPS = **Compare String** (Find Match)	1111 0010 : 1010 011w			
C = 0		5		16, 17
C > 0		7+7c		
REP LODS = **Load String**	1111 0010 : 1010 110w			
C = 0		5		16, 18
C > 0		7+4c		
REP MOVS = **Move String**	1111 0010 : 1010 010w			
C = 0		5	1	16
C = 1		13		16, 19
C > 1		12+3c		
REPE SCAS = **Scan String** (Find Non-AL/AX/EAX)	1111 0011 : 1010 111w			
C = 0		5		
C > 0		7+5c		20
REPNE SCAS = **Scan String** (Find AL/AX/EAX)	1111 0010 : 1010 111w			
C = 0		5		
C > 0		7+5c		20
REP STOS = **Store String**	1111 0010 : 1010 101w			
C = 0		5		
C > 0		7+4c		

Table 13-15. Clock Count Summary (Continued)

Instruction	Format	Cache Hit	Penalty if Cache Miss	Notes
FLAG CONTROL				
CLC = Clear Carry Flag	1111 1000	2		
STC = Set Carry Flag	1111 1001	2		
CMC = Complement Carry Flag	1111 0101	2		
CLD = Clear Direction Flag	1111 1100	2		
STD = Set Direction Flag	1111 1101	2		
CLI = Clear Interrupt Enable Flag	1111 1010	5		
STI = Set Interrupt Enable Flag	1111 1011	5		
LAHF = Load AH into Flag	1001 1111	3		
SAHF = Store AH into Flag	1001 1110	2		
PUSHF = Push Flags	1001 1100	4/3		RV/P
POFF = Pop Flags	1001 1101	9/6		RV/P
DECIMAL ARITHMETIC				
AAA = ASCII Adjust to Add	0011 0111	3		
AAS = ASCII Adjust for Subtract	0011 1111	3		
AAM = ASCII Adjust for Multiply	1101 0100 : 0000 1010	15		
AAD = ASCII Adjust for Divide	1101 0101 : 0000 1010	14		
DAA = Decimal Adjust for Add	0010 0111	2		
DAS = Decimal Adjust for Subtract	0010 1111	2		
PROCESSOR CONTROL INSTRUCTIONS				
HLT = Halt	1111 0100	4		
MOV = Move To and From Control/Debug/Test Registers				
CR0 from register	0000 1111 : 0010 0010 : 11 000 reg	17	2	
CR2/CR3 from register	0000 1111 : 0010 0010 : 11 eee reg	4		
Reg from CR0-3	0000 1111 : 0010 0000 : 11 eee reg	4		
DR0-3 from register	0000 1111 : 0010 0011 : 11 eee reg	10		
DR6-7 from register	0000 1111 : 0010 0011 : 11 eee reg	10		
Register from DR6-7	0000 1111 : 0010 0001 : 11 eee reg	9		
Register from DR0-3	0000 1111 : 0010 0001 : 11 eee reg	9		
TR3 from register	0000 1111 : 0010 0110 : 11 011 reg	4		
TR4-7 from register	0000 1111 : 0010 0110 : 11 eee reg	4		
Register from TR3	0000 1111 : 0010 0100 : 11 011 reg	3		
Register from TR4-7	0000 1111 : 0010 0100 : 11 eee reg	4		

Table 13-15. Clock Count Summary (Continued)

Instruction	Format	Cache Hit	Penalty if Cache Miss	Notes
PROCESSOR CONTROL INSTRUCTIONS (Continued)				
CPUID = CPU Identification	0000 1111 : 1010 0010			
EAX = 1		14		
EAX = 0, >1		9		
CLTS = Clear Task Switched Flag	0000 1111 : 0000 0110	7	2	
INVD = Invalidate Data Cache	0000 1111 : 0000 1000	4		
WBINVD = Write-Back and Invalidate Data Cache	0000 1111 : 0000 1001	5		
INVLPG = Invalidate TLB Entry				
INVLPG memory	0000 1111 : 0000 0001 : mod 111 r/m	12/11		H/NH
PREFIX BYTES				
Address Size Prefix	0110 0111	1		
LOCK = Bus Lock Prefix	1111 0000	1		
Operand Size Prefix	0110 0110	1		
Segment Override Prefix				
CS:	0010 1110	1		
DS:	0011 1110	1		
ES:	0010 0110	1		
FS:	0110 0100	1		
GS:	0110 0101	1		
SS:	0011 0110	1		
PROTECTION CONTROL				
ARPL = Adjust Requested Privilege Level				
From register	0110 0011 : 11 reg1 reg2	9		
From memory	0110 0011 : mod reg r/m	9		
LAR = Load Access Rights				
From register	0000 1111 : 0000 0010 : 11 reg1 reg2	11	3	
From memory	0000 1111 : 0000 0010 : mod reg r/m	11	5	
LGDT = Load Global Descriptor				
Table register	0000 1111 : 0000 0001 : mod 010 r/m	12	5	
LIDT = Load Interrupt Descriptor				
Table register	0000 1111 : 0000 0001 : mod 011 r/m	12	5	
LLDT = Load Local Descriptor				
Table register from reg.	0000 1111 : 0000 0000 : 11 010 reg	11	3	
Table register from mem.	0000 1111 : 0000 0000 : mod 010 r/m	11	6	

Table 13-15. Clock Count Summary (Continued)

Instruction	Format	Cache Hit	Penalty if Cache Miss	Notes
PROTECTION CONTROL (Continued)				
LMSW = Load Machine Status Word				
From register	0000 1111 : 0000 0001 : 11 110 reg	13		
From memory	0000 1111 : 0000 0001 : mod 110 r/m	13	1	
LSL = Load Segment Limit				
From register	0000 1111 : 0000 0011 : 11 reg1 reg2	10	3	
From memory	0000 1111 : 0000 0011 : mod reg r/m	10	6	
LTR = Load Task Register				
From register	0000 1111 : 0000 0000 : 11 011 reg	20		
From memory	0000 1111 : 0000 0000 : mod 011 r/m	20		
SGDT = Store Global Descriptor Table	0000 1111 : 0000 0001 : mod 000 r/m	10		
SIDT = Store Interrupt Descriptor Table	0000 1111 : 0000 0001 : mod 001 r/m	2		
SLDT = Store Local Descriptor Table				
To register	0000 1111 : 0000 0000 : 11 000 reg	2		
To memory	0000 1111 : 0000 0001 : mod 000 r/m	3		
SMSW = Store Machine Status Word				
To register	0000 1111 : 0000 0001 : 11 000 reg	2		
To memory	0000 1111 : 0000 0001 : mod 100 r/m	3		
STR = Store Task Register				
To register	0000 1111 : 0000 0000 : 11 001 r/m	2		
To memory	0000 1111 : 0000 0000 : mod 001 r/m	3		
VERR = Verify Read Access				
Register	0000 1111 : 0000 0000 : 11 100 r/m	11	3	
Memory	0000 1111 : 0000 0000 : mod 100 r/m	11	7	
VERW = Verify Write Access				
To register	0000 1111 : 0000 0000 : 11 101 r/m	11	3	
To memory	0000 1111 : 0000 0000 : mod 101 r/m	11	7	
INTERRUPT INSTRUCTIONS				
INTn = Interrupt Type n	1100 1101 : type	INT + 4/0		RV/P, 21
INT3 = Interrupt Type 3	1100 1100	INT + 0		21

Table 13-15. Clock Count Summary (Continued)

Instruction	Format	Cache Hit	Penalty if Cache Miss	Notes
INTERRUPT INSTRUCTIONS (Continued)				
INTO = Interrupt 4 if Overflow Flag Set	1100 1110			
Taken		INT + 2		21
Not Taken		3		21
BOUND = Interrupt 5 if Detect Value Out Range	0110 0010 : mod reg r/m			
If in range		7	7	21
If out of range		INT + 24	7	21
IRET = Interrupt Return	1100 1111			
Real Mode/Virtual Mode		15		
Protected Mode				
To same level		20	11	9
To outer level		36	19	9
To nested task		TS + 32	4	9,10
(EFLAGS.NT = 1)				
RSM = Exit System Management Mode	0000 1111 : 1010 1010			
SMBASE Relocation		452		
Auto HALT Restart		456		
I/O Trap Restart		465		
External Interrupt		INT + 11		21
NMI = Non-Maskable Interrupt		INT + 3		21
Page Fault		INT + 24		21
VM86 Exceptions				
CLI		INT + 8		21
STI		INT + 8		21
INTn		INT + 9		
PUSHF		INT + 9		21
POPF		INT + 8		21
IRET		INT + 9		
IN				
Fixed Port		INT + 50		21
Variable Port		INT + 51		21
OUT				
Fixed Port		INT + 50		21
Variable Port		INT + 51		21
INS		INT + 50		21
OUTS		INT + 50		21
REP INS		INT + 51		21
REP OUTS		INT + 51		21

Table 13-16. Task Switch Clock Counts

Method	Value for TS	
	Cache Hit	Miss Penalty
VM/Military Intel486 Processor/286 TSS to Military Intel486 Processor TSS	162	55
VM/Military Intel486 Processor/286 TSS to 286 TSS	144	31
VM/Military Intel486 Processor/286 TSS to VM TSS	140	37

Table 13-17. Interrupt Clock Counts

Method	Value for INT		
	Cache Hit	Miss Penalty	Notes
Real Mode	26	2	
Protected Mode			
Interrupt/Trap gate, same level	44	6	9
Interrupt/Trap gate, different level	71	17	9
Task Gate	37 + TS	3	9, 10
Virtual Mode			
Interrupt/Trap gate, different level	82	17	
Task Gate	37 + TS	3	10

Abbreviations	Definition
16/32	16/32 bit modes
U/L	unlocked/locked
MN/MX	minimum/maximum
L/NL	loop/no loop
RV/P	real and virtual mode/protected mode
R	real mode
P	protected mode
T/NT	taken/not taken
H/NH	hit/no hit

NOTES (for Tables 13-17 through 13-19):

1. Assuming that the operand address and stack address fall in different cache sets.
2. Always locked, no cache hit case.
3. Clocks = 10 + max($\log_2$(|m|),n)
4. Clocks = {qoutient(couint/operand length)}*7+9
   = 8 if count ≤ operand length (8/16/32)
5. Clocks = {qoutient(couint/operand length)}*7+9
   = 9 if count ≤ operand length (8/16/32)
6. Equal/not equal cases (penalty is the same regardless of lock)
7. Assuming that addresses for memory read (for indirection), stack puch/pop and branch fall in different cache sets.
8. Penalty for cache miss: add 6 clocks for every 16 bytes copied to new stack frame.
9. Add 11 clocks for each unaccessed descriptor load.
10. Refer to task switch clock counts table for value of TS.
11. Add 4 extra clocks to the cache miss penalty for each 16 bytes.

For notes 12-13: (b = 0-3, non-zero byte number);
(i = 0-1, non-zero nibble number);
(n = 0-3, non-bit number in nibble);
12. Clocks = 8 + 4 (b + 1) + 3(i + 1) + 3(n + 1)
= 6 if second operand = 0
13. Clocks = 9 + 4 (b + 1) + 3(i + 1) + 3(n + 1)
= 7 if second operand = 0
For notes 14-15: (n = bit position 0-31)
14. Clocks = 7 + 3(32-n)
= 6 if second operand = 0
15. Clocks = 8 + 3(32-n)
= 7 if second operand = 0
16. Assuming that the two string addresses fall in different cache sets.
17. Cache miss penalty: add 6 clocks for every 16 bytes compared. Entire penalty on first compare.
18. Cache miss penalty: add 2 clocks for every 16 bytes of data. Entire penalty on first load.
19. Cache miss penalty: add 4 clocks for every 16 bytes moved. (1 clock for the first operation and 3 for the second)
20. Cache miss penalty: add 4 clocks for every 16 bytes scanned. (2 clocks each for first and second operations)
21. Refer to interrupt clock counts table for value of INT.
22. Clock count includes one clock for using both displacement and immediate.
23. Refer to assumption 6 in the case of a cache miss.
24. Virtual Mode Extensions are disabled.
25. Protected Virtual Interrupts are disabled.

Table 13-18. I/O Instructions Clock Count Summary

Instruction	Format	Real Mode	Protected Mode (CPL ≤ IOPL)	Protected Mode (CPL > IOPL)	Virtual 86 Mode	Notes
IN = Input from:						
Fixed Port	1110 010w : port number	14	9	29	27	
Variable Port	1110 110w	14	8	28	27	
OUT = Output to:						
Fixed Port	1110 011w : port number	16	11	31	29	
Variable Port	1110 110w	16	10	30	29	
INS = Input Byte/Word from DX Port	0110 110w	17	10	32	30	
OUTS = Output Byte/Word to DX Port	0110 111w	17	10	32	30	1
REP INS = Input String	1111 0010 : 0110 110w	16 + 8c	10 + 8c	30 + 8c	29 + 8c	2
REP OUTS = Output String	1111 0010 : 0110 111w	17 + 5c	11 + 5c	31 + 5c	30 + 5c	3

NOTES:
1. Two clock cache miss penalty in all cases.
2. c = count in CX or ECX.
3. Cache miss penalty in all modes: Add 2 clocks for every 16 bytes. Entire penalty on second operation.

Table 13-19. Floating Point Clock Count Summary

Instruction	Format	Cache Hit: Avg (Lower Range . . . Upper Range)	Penalty if Cache Miss	Concurrent Execution: Avg (Lower Range . . . Upper Range)	Notes
DATA TRANSFER					
FLD = Real Load to ST(0)					
32-bit memory	11011 001 : mod 000 r/m : s-i-b/disp.	3	2		
64-bit memory	11011 101 : mod 000 r/m : s-i-b/disp.	3	3		
80-bit memory	11011 011 : mod 101 r/m : s-i-b/disp.	6	4		
ST(i)	11011 001 : 11000 ST(i)	4			
FILD = Integer Load to ST(0)					
16-bit memory	11011 111 : mod 000 r/m : s-i-b/disp.	14.5(13-16)	2	4	
32-bit memory	11011 011 : mod 000 r/m : s-i-b/disp.	11.5(9-12)	2	4(2-4)	
64-bit memory	11011 111 : mod 101 r/m : s-i-b/disp.	16.8(10-18)	3	7.8(2-8)	
FBLD = BCD Load to ST(0)					
	11011 111 : mod 100 r/m : s-i-b/disp.	75(70-103)	4	7.7(2-8)	
FST = Store Real from ST(0)					
32-bit memory	11011 011 : mod 010 r/m : s-i-b/disp.	7			1
64-bit memory	11011 101 : mod 010 r/m : s-i-b/disp.	8			2
ST(i)	11011 101 : 11001 ST(i)	3			
FSTP = Store Real from ST(0) and Pop					
32-bit memory	11011 011 : mod 011 r/m : s-i-b/disp.	7			1
64-bit memory	11011 101 : mod 011 r/m : s-i-b/disp.	8			2
80-bit memory	11011 011 : mod 111 r/m : s-i-b/disp.	6			
ST(i)	11011 101 : 11001 ST(i)	3			
FIST = Store Integer from ST(0)					
16-bit memory	11011 111 : mod 010 r/m : s-i-b/disp.	33.4(29-34)			
32-bit memory	11011 011 : mod 010 r/m : s-i-b/disp.	32.4(28-34)			
FISTP = Store Integer from ST(0) and Pop					
16-bit memory	11011 111 : mod 011 r/m : s-i-b/disp.	33.4(29-34)			
32-bit memory	11011 011 : mod 011 r/m : s-i-b/disp.	33.4(29-34)			
64-bit memory	11011 111 : mod 111 r/m : s-i-b/disp.	33.4(29-34)			
FBSTP = Store BCD from ST(0) and Pop					
	11011 111 : mod 110 r/m : s-i-b/disp.	175(172-176)			
FXCH = Exchange ST(0) and ST(i)					
	11011 001 : 11001 ST(i)	4			

Table 13-19. Floating Point Clock Count Summary (Continued)

Instruction	Format	Cache Hit: Avg (Lower Range . . . Upper Range)	Penalty if Cache Miss	Concurrent Execution: Avg (Lower Range . . . Upper Range)	Notes
COMPARISON INSTRUCTIONS					
FCOM = Compare ST(0) with Real					
32-bit memory	11011 000 : mod 010 r/m : s-i-b/disp.	4	2	1	
64-bit memory	11011 100 : mod 010 r/m : s-i-b/disp.	4	3	1	
ST(i)	11011 000 : 11010 ST(i)	4			
FCOMP = Compare ST(0) with Real and Pop					
32-bit memory	11011 000 : mod 011 r/m : s-i-b/disp.	4	2	1	
64-bit memory	11011 100 : mod 011 r/m : s-i-b/disp.	4	3	1	
ST(i)	11011 000 : 11011 ST(i)	4		1	
FCOMPP = Compare ST(0) with ST(1) and Pop Twice					
	11011 110 : 1101 1001	5		1	
FICOM = Compare ST(0) with Integer					
16-bit memory	11011 110 : mod 010 r/m : s-i-b/disp.	18(16-20)	2	1	
32-bit memory	11011 010 : mod 010 r/m : s-i-b/disp.	16.5(15-17)	2	1	
FICOMP = Compare ST(0) with Integer					
16-bit memory	11011 110 : mod 011 r/m : s-i-b/disp.	18(16-20)	2	1	
32-bit memory	11011 010 : mod 011 r/m : s-i-b/disp.	16.5(15-17)	2	1	
FTST = Compare ST(0) with 0.0					
	11011 011 : 1110 0100	4		1	
FUCOM = Unordered compare ST(0) with ST(i)					
	11011 101 : 11100 ST(i)	4		1	
FUCOMP = Unordered compare ST(0) with ST(i) and Pop					
	11011 101 : 11101 ST(i)	4		1	
FUCOMPP = Unordered compare ST(0) with ST(1) and Pop Twice					
	11011 101 : 11101 1001	5		1	
FXAM = Examine ST(0)					
	11011 001 : 1110 0101	8			

Table 13-19. Floating Point Clock Count Summary (Continued)

Instruction Format	Cache Hit: Avg (Lower Range . . . Upper Range)	Penalty if Cache Miss	Concurrent Execution: Avg (Lower Range . . . Upper Range)	Notes
CONSTANTS				
FLDZ = Load +0.0 Into ST(0) 11011 001 : 1110 1110 :	4			
FLD1 = Load +1.0 Into ST(0) 11011 001 : 1110 1000 :	4			
FLDP1 = Load π Into ST(0) 11011 001 : 1110 1011 :	8		2	
FLDL2T = Load $\log_2(10)$ Into ST(0) 11011 001 : 1110 1001 :	8		2	
FLDL2E = Load $\log_2(e)$ Into ST(0) 11011 001 : 1110 1010 :	8		2	
FLDLG2 = Load $\log_{10}(2)$ Into ST(0) 11011 001 : 1110 1100 :	8		2	
FLDLN2 = Load $\log_e(2)$ Into ST(0) 11011 001 : 1110 1101 :	8		2	
ARITHMETIC				
FADD = Add Real with ST(0)				
ST(0) ← ST(0) + 32-bit memory 11011 000 : mod 000 r/m : s-i-b/disp.	10(8-20)	2	7(5-17)	
ST(0) ← ST(0) + 64-bit memory 11011 100 : mod 000 r/m : s-i-b/disp.	10(8-20)	3	7(5-17)	
ST(d) ← ST(0) + ST(i) 11011 d00 : 11000 ST(i)	10(8-20)		7(5-17)	
FADDP = Add real with ST(0) and Pop (ST(i) ← ST(0) +ST(i)) 11011 110 : 11000 ST(i) :	10(8-20)		7(5-17)	

Table 13-19. Floating Point Clock Count Summary (Continued)

Instruction Format	Cache Hit Avg (Lower Range . . . Upper Range)	Penalty if Cache Miss	Concurrent Execution Avg (Lower Range . . . Upper Range)	Notes
ARITHMETIC (Continued)				
FSUB = Subtract Real from ST(0)				
ST(0) ← ST(0) − 32-bit memory				
11011 000 : mod 100 r/m : s-i-b/disp.	10(8-20)	2	7(5-17)	
ST(0) ← ST(0) − 64-bit memory				
11011 100 : mod 100 r/m : s-i-b/disp.	10(8-20)	3	7(5-17)	
ST(d) ← ST(0) − ST(i)				
11011 d00 : 11001 ST(i)	10(8-20)		7(5-17)	
FSUBP = Subtract real from ST(0) and Pop (ST(i) ← ST(0) − ST(i))				
11011 110 : 11001 ST(i)	10(8-20)		7(5-17)	
FSUBR = Subtract Real reversed (Subtract ST(0) from Real)				
ST(0) ← 32-bit memory − ST(0)				
11011 000 : mod 101 r/m : s-i-b/disp.	10(8-20)	2	7(5-17)	
ST(0) ← 64-bit memory − ST(0)				
11011 100 : mod 101 r/m : s-i-b/disp.	10(8-20)	3	7(5-17)	
ST(d) ← ST(i) − ST(0)				
11011 d00 : 11001 ST(i)	10(8-20)		7(5-17)	
FSUBRP = Subtract Real reversed and Pop (ST(i) ← ST(i) − ST(0))				
11011 110 : 11100 ST(i)	10(8-20)		7(5-17)	
FMUL = Multiply Real with ST(0)				
ST(0) ← ST(0) X 32-bit memory				
11011 000 : mod 001 r/m : s-i-b/disp.	11	2	8	
ST(0) ← ST(0) X 64-bit memory				
11011 100 : mod 001 r/m : s-i-b/disp.	14	3	11	
ST(d) ← ST(0) X ST(i)				
11011 d00 : 11001 ST(i)	16		13	
FMULP = Multiply ST(0) with ST(i) and Pop (ST(i) ← ST(0) XST(i))				
11011 110 : 11001 ST(i)	16		13	
FDIV = Divide ST(0) by Real				
ST(0) ← ST(0)/ 32-bit memory				
11011 000 : mod 110 r/m : s-i-b/disp.	73	2	70	3
ST(0) ← ST(0)/ 64-bit memory				
11011 100 : mod 110 r/m : s-i-b/disp.	73	3	70	3
ST(d) ← ST(0)/ ST(i)				
11011 d00 : 11111 ST(i)	73		70	3
FDIVP = Divide ST(0) by ST(i) and Pop (ST(i) ← ST(0)/ ST(i))				
11011 110 : 11111 ST(i)	73		70	3

Table 13-19. Floating Point Clock Count Summary (Continued)

Instruction Format	Cache Hit: Avg (Lower Range . . . Upper Range)	Penalty if Cache Miss	Concurrent Execution: Avg (Lower Range . . . Upper Range)	Notes
ARITHMETIC (Continued)				
FDIVR = Divide real reversed (Real/ST(0))				
ST(0) ← 32-bit memory/ ST(0)				
11011 000 : mod 111 r/m : s-i-b/disp.	73	2	70	3
ST(0) ← 64-bit memory/ ST(0)				
ST(d) ← ST(i)/ ST(0) 11011 100 : mod 111 r/m : s-i-b/disp.	73	3	70	3
11011 d00 : 11110 ST(i)	73		70	3
FDIVRP = Divide real reversed and Pop (ST(i) ← ST(i)/ ST(0))				
11011 110 : 11110 ST(i)	73		70	3
FIADD = Add Integer to ST(0)				
ST(0) ← ST(0) + 16-bit memory				
11011 110 : mod 000 r/m : s-i-b/disp.	24(20-35)	2	7(5-17)	
ST(0) ← ST(0) + 32-bit memory				
11011 010 : mod 000 r/m : s-i-b/disp.	22.5(19-32)	2	7(5-17)	
FISUB = Subtract Integer from ST(0)				
ST(0) ← ST(0) − 16-bit memory				
11011 110 : mod 100 r/m : s-i-b/disp.	24(20-35)	2	7(5-17)	
ST(0) ← ST(0) − 32-bit memory				
11011 010 : mod 100 r/m : s-i-b/disp.	22.5(19-32)	2	7(5-17)	
FISUBR = Integer Subtract Reversed				
ST(0) ← 16-bit memory − ST(0)				
11011 110 : mod 101 r/m : s-i-b/disp.	24(20-35)	2	7(5-17)	
ST(0) ← 32-bit memory − ST(0)				
11011 010 : mod 101 r/m : s-i-b/disp.	22.5(19-32)	2	7(5-17)	
FIMUL = Multiply Integer with ST(0)				
ST(0) ← ST(0) X 16-bit memory				
11011 110 : mod 101 r/m : s-i-b/disp.	25(23-27)	2	8	
ST(0) ← ST(0) X 32-bit memory				
11011 010 : mod 001 r/m : s-i-b/disp.	23.5(19-32)	2	8	

Table 13-19. Floating Point Clock Count Summary (Continued)

Instruction Format	Cache Hit Avg (Lower Range . . . Upper Range)	Penalty if Cache Miss	Concurrent Execution Avg (Lower Range . . . Upper Range)	Notes
ARITHMETIC (Continued)				
FIDIV = Integer Divide				
ST(0) ← ST(0)/ 16-bit memory				
11011 110 : mod 110 r/m : s-i-b/disp.	87(85-89)	2	70	3
ST(0) ← ST(0)/ 32-bit memory				
11011 010 : mod 110 r/m : s-i-b/disp.	85.5(84-86)	2	70	3
FIDVR = Integer Divide Reversed				
ST(0) ← 16-bit memory/ST(0)				
11011 110 : mod 111 r/m : s-i-b/disp.	87(85-89)	2	70	3
ST(0) ← 32-bit memory/ST(0)				
11011 010 : mod 111 r/m : s-i-b/disp.	85.5(84-86)	2	70	3
FSQRT = Square Root				
11011 001 : 1111 1010	85.5(83-87)		70	
FSCALE = Scale ST(0) by ST(1)				
11011 001 : 1111 1101	31(30-32)		2	
FXTRACT = Extract Components of ST(0)				
11011 001 : 1111 0100	19(16-20)		4(2-4)	
FPREM = Partial Reminder				
11011 001 : 1111 1000	84(70-138)		2(2-8)	
FPREM1 = Partial Reminder (IEEE)				
11011 001 : 1111 0101	94.5(72-167)		5.5(2-18)	
FRNDINT = Round ST(0) to Integer				
11011 001 : 1111 1100	29.1(21-30)		7.4(2-8)	
FABS = Absolute value of ST(0)				
11011 001 : 1110 0001	3			
FCHS = Change Sign of ST(0)				
11011 001 : 1110 0000	6			
TRANSCENDENTAL				
FCOS = Cosine of ST(0)				
11011 001 : 1111 1111	241(193-279)		2	6,7
FPTAN = Partial Tangent of ST(0)				
11011 001 : 1111 0010	244(200-273)		70	6,7
FPATAN = Partial Arctangent				
11011 001 : 1111 0011	289(218-303)		5(2-17)	6

Table 13-19. Floating Point Clock Count Summary (Continued)

Instruction Format	Cache Hit: Avg (Lower Range . . . Upper Range)	Penalty if Cache Miss	Concurrent Execution: Avg (Lower Range . . . Upper Range)	Notes
TRANSCENDENTAL (Continued)				
FSIN = Sine of ST(0) 11011 001 : 1111 1110	241(193-279)		2	6,7
FSINCOS = Sine and Cosine of ST(0) 11011 001 : 1111 1011	291(243-329)		2	6,7
F2XM1 = $2^{ST(0)}-1$ 11011 001 : 1111 0000	242(140-279)		2	6
FYL2X = ST(1) x $\log_2$(ST(0)) 11011 001 : 1111 0001	311(196-329)		13	6
FYL2XP1 = ST(1) x $\log_2$(ST(0) + 1.0) 11011 001 : 1111 1001	313(171-326)		13	6
PROCESSOR CONTROL				
FINIT = Initialize FPU 11011 001 : 1110 0011	17			4
FSTSW AX = Store status word into AX 11011 111 : 1110 0000	3			5
FSTSW = Store status word into memory 11011 101 : mod 111 r/m : s-i-b/disp.	3			5
FLDCW = Load control word 11011 001 : mod 101 r/m : s-i-b/disp.	4	2		
FSTCW = Store control word 11011 001 : mod 111 r/m : s-i-b/disp.	3			5
FCLEX = Clear exceptions 11011 011 : 1110 0010	7			4
FSTENV = Store environment 11011 011 : mod 110 r/m : s-i-b/disp.				
Real and Virtual Modes 16-bit address	67			4
Real and Virtual Modes 32-bit address	67			4
Protected Mode 16-bit address	56			4
Protected Mode 32-bit address	56			4
FLDENV = Load Environment 11011 011 : mod 100 r/m : s-i-b/disp.				
Real and Virtual Modes 16-bit address	44	2		
Real and Virtual Modes 32-bit address	44	2		
Protected Mode 16-bit address	34	2		
Protected Mode 32-bit address	34	2		

Table 13-19. Floating Point Clock Count Summary (Continued)

Instruction Format	Cache Hit Avg (Lower Range . . . Upper Range)	Penalty if Cache Miss	Concurrent Execution Avg (Lower Range . . . Upper Range)	Notes
PROCESSOR CONTROL (Continued)				
FSAVE = Save State				
11011 101 : mod 110 r/m : s-i-b/disp.				
Real and Virtual Modes 16-bit address	154			4
Real and Virtual Modes 32-bit address	154			4
Protected Mode 16-bit address	143			4
Protected Mode 32-bit address	143			4
FRSTOR = Restore State				
11011 101 : mod 100 r/m : s-i-b/				
Real and Virtual Modes 16-bit address	131	23		
Real and Virtual Modes 32-bit address	131	27		
Protected Mode 16-bit address	120	23		
Protected Mode 32-bit address	120	27		
FINCSTP = Increment Stack Pointer				
11011 001 : 1111 0111	3			
FDECSTP = Decrement Stack Pointer				
11011 001 : 1111 0110	3			
FFREE = Free ST(i)				
11011 101 : 11000 ST(i)	3			
FNOP = No Operations				
11011 101 : 1101 0000	3			
WAIT = Wait until FPU ready (min/max)				
10011011	1⁄3			

NOTES:
1. If operand is 0 clock counts = 27.
2. If operand is 0 clock counts = 28.
3. If CW.PC indicates 24-bit precision then subtract 38 clocks.
   If CW.PC indicates 53-bit precision then subtract 11 clocks.
4. If there is a numeric error pending from a previous instruction add 17 clocks.
5. If there is a numeric error pending from a previous instruction add 18 clocks.
6. The INT pin is polled several times while this function is executing to assure short interrupt latency.
7. If ABS(operand) is greater than $\pi/4$ then add n clocks, where n = (operand/($\pi/4$)).